

VECPAR 98

**3rd internacional meeting on
vector and parallel processing**

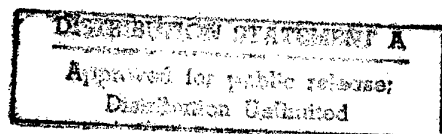
**1998
June, 21 - 23**



**Faculdade de Engenharia
da Universidade do Porto**



**Proceedings
Part III (June 23)**



19980925 007

THIS QUALITY INSPECTED 1

AQF98-12-2594

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE 10 August 1998		3. REPORT TYPE AND DATES COVERED Conference Proceedings	
4. TITLE AND SUBTITLE VECPAR 98 3rd International Meeting on Vector and Parallel Processing				5. FUNDING NUMBERS F6170898W0009	
6. AUTHOR(S) Conference Committee					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Faculdade de Engenharia da Universidade do Porto Seccao dos Bragas Porto Codex 4099 Portugal				8. PERFORMING ORGANIZATION REPORT NUMBER N/A	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) EOARD PSC 802 BOX 14 FPO 09499-0200				10. SPONSORING/MONITORING AGENCY REPORT NUMBER CSP 98-1006	
11. SUPPLEMENTARY NOTES Consists of three volumes.					
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.				12b. DISTRIBUTION CODE A	
13. ABSTRACT (Maximum 200 words) The Final Proceedings for VECPAR 98 3rd International Meeting on Vector and Parallel Processing, 21 June 1998 - 23 June 1998 This is an interdisciplinary conference. Topics include parallel and distributed computing, image processing and synthesis, real-time and embedded systems.					
14. SUBJECT TERMS Computers, Signal Processing, Mathematics, Modelling & Simulation				15. NUMBER OF PAGES 1088	
				16. PRICE CODE N/A	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL		

VECPAR'98

3rd International Meeting on
Vector and Parallel Processing

1998, June 21-23

Conference Proceedings

Part III

(Tuesday, June 23)



FEUP

Faculdade de Engenharia
da Universidade do Porto

Table of Contents

PART I

Invited Talk 1

- *Some Unusual Eigenvalue Problems* 1
Zhajun Bai and Gene Golub (USA)

Technical Session 1

- *Parallel Preconditioners for Solving Nonsymmetric Linear Systems* 17
Antonio J. García-Loureiro, Tomás F. Pena, J.M. López-González and Ll. Prat Viñas (Spain)
- *Parallel Preconditioned Solvers for Large Sparse Hermitian Eigenproblems* 31
A. Basermann (Germany)
- *Comparisons of Parallel Algorithms to Evaluate Orthogonal Series* 45
R. Barrio (Spain)

Technical Session 2

- *Coarse-grain Parallelization of a Multi-Block Navier-Stokes Solver on a Shared Memory Parallel Vector Computer* 59
P. Wijnandts and M.E.S. Vogels (The Netherlands)
- *Using Synthetic Workloads for Parallel Task Scheduling Improvement Analysis* 73
João Paulo Kitajima and Stella Porto (Brazil)
- *Influence of the Discretization Scheme on the Parallel Efficiency of a Code for the Modelling of a Utility Boiler* 87
P.J. Coelho (Portugal)

Technical Session 3

- *Parallel Implementation of Edge-Based Finite Element Schemes for Compressible Flows on Unstructured Grids* 99
P.R.M. Lyra, R.B. Willmersdorf, M.A.D. Martins and A.L.G.A. Coutinho (Brazil)

• <i>Parallel 3D Air Flow Simulation on Workstation Cluster</i> Jean-Baptiste Vicaire, Loic Prylli, Georges Perrot and Bernard Tourancheau (France)	113
• <i>2D Pseudo-Spectral Parallel Navier-Stokes Simulations of the Rayleigh-Taylor Instability</i> E. Fournier and S. Gauthier (France)	127
Technical Session 4	
• <i>A Unified Approach to Parallel Block-Jacobi Methods for the Symmetric Eigenvalue Problem</i> D. Giménez, V. Hernández and A. M. Vidal (Spain)	139
• <i>Solving Large-Scale Eigenvalue Problems on Vector-Parallel Processors</i> David L. Harrar II and Michael R. Osborne (Australia)	153
• <i>Solving Eigenvalue Problems on Networks of Processors</i> D. Giménez, C. Jiménez, M., J. Majado, N. Marín and A. Martín (Spain)	167
Invited Talk 2	
• <i>Parallel Domain-Decomposition Preconditioning for Computational Fluid Dynamics</i> Timothy Barth, Tony Chan and Wei-Pai Tang (USA)	181
Technical Session 5	
• <i>Parallel Turbulence Simulation: Resolving the Inertial Subrange of Kolmogorov's Spectra</i> Thomas Gerz and Martin Strietzel (Germany)	209
• <i>A Systolic Algorithm for the Factorisation of Matrices Arising in the Field of Hydrodynamics</i> S. G. Seo, M. J. Downie, G. E. Hearn and C. Phillips (UK)	217
• <i>The Study of a Parallel Algorithm Using the Backward-Facing Step Flow as a Test Case</i> P.M. Areal and J.M.L.M. Palma (Portugal)	227
• <i>High Performance Cache Management for Parallel File Systems</i> F. García, J. Carretero, F. Pérez and P. de Miguel (Spain)	239

Technical Session 6

- *Parallel Jacobi-Davidson for Solving Generalized Eigenvalue Problems* 253
Margreet Nool and Auke van der Ploeg (The Netherlands)
- *A Level 3 Algorithm for the Symmetric Eigenproblem* 267
Dieter F. Kvasnicka, Wilfried N. Gansterer and Christoph W. Ueberhuber (Austria)
- *Synchronous and Asynchronous Parallel Algorithms with Overlap for Almost Linear Systems* 277
Josep Arnal, Violeta Migallón and José Penadés (Spain)
- *Spatial Data Locality With Respect to Degree of Parallelism in Processor-and-Memory Hierarchies* 291
Renato J. O. Figueiredo, José A. B. Fortes and Zina Ben Miled (USA)

PART II

Technical Session 7

- *Partitioning Regular Domains on Modern Parallel Computers* 305
M. Prieto-Matías, I. Martín-Llorente and F. Tirado-Fernández (Spain)
- *A Performance Analysis of the SGI Origin2000* 319
Aad J. van der Steen and Ruud van der Pas (The Netherlands)
- *Parallel Computing over the Internet with Java* 333
Hernâni Pedroso, Luís M. Silva, Vítor Batista, Paulo Martins, Guilherme Soares and Telmo Menezes (Portugal)
- *The Parallel Problems Server: A Client-Server Model for Interactive Large Scale Scientific Computation* 345
Parry Husbands and Charles L. Isbell (USA)

Technical Session 8

- *A Thread-level Distributed Debugger* 359
João Lourenço and José C. Cunha (Portugal)
- *New Access Order to Reduce Inter-Vector Conflicts* 367
A. M. del Corral and J. M. Llabetria (Spain)
- *Multilevel Mesh Partitioning for Aspect Ratio* 381
C. Walshaw, M. Cross, R. Diekmann and F. Shlimbach (UK)

- *Visualization of HPF Data Mappings and of their Communication Cost* 395
Christian Lefebvre and Jean-Luc Dekeyser (France)

Invited Talk 3

- *Parallel and Distributed Computing in Education* 409
Peter Welch (UK)

Technical Session 9

- *An ISA comparison between Superscalar and Vector Processors* 439
Francisca Quintana, Roger Espasa and Mateo Valero (Spain)
- *Implementing the Time-Warp Simulation Model in Java* 453
Pedro Bizarro, Luís M. Silva and João Gabriel Silva (Portugal)
- *Evaluation of High Performance Fortran for an Industrial Computational Fluid Dynamics Code* 467
Thomas Brandes, Falk Zimmermann, Christian Borel and Marc Brédif (Germany)

Technical Session 10

- *Automatic Detection of Parallel Program Performance Problems* 481
Antonio Espinosa, Tomàs Margalef and Emilio Luque (Spain)
- *Registers Size Influence on Vector Architectures* 495
Luis Villa, Roger Espasa and Mateo Valero (Spain)
- *The Adaptive Restarted Procedure for ORTHOMIN(k) Algorithm* 507
Takashi Nodera and Naoto Tsuno (Japan)

Invited Talk 4

- *Reconfigurable Systems: Past and Next 10 Years* 519
Jean Vuillemin (France)

Technical Session 11

- *A Method Based on Orthogonal Transformation for the Design of Optimal Feedforward Network Architecture* 541
Bachiller P., Pérez R.M., Martinez P., Aguilar P.L., Calle J.E. (Spain)
- *Preprocessor Based Implementation of the Versatile Advection Code for Workstations, Vector and Parallel Computers* 553
Gábor Tóth (Hungary)

- *A Parallel N-Body Integrator Using MPI* 561
Nuno S. A. Pereira (Portugal)
- *Efficient Molecular Dynamics on a Network of Personal Computers* 575
Giuseppe Ciaccio and Vincenzo Di Martino (Italy)

Technical Session 12

- *Limits of Instruction Level Parallelism with Data Speculation* 585
José González and Antonio González (Spain)
- *Simulating Magnetized Plasma with the Versatile Advection Code* 599
R. Keppens and G. Tóth (The Netherlands)
- *Parallel Grid Manipulations in Earth Science Calculations* 611
W. Sawyer, L. L. Takacs, A. da Silva, P. M. Lyster (USA)
- *Molecular Dynamics as a Natural Solver* 625
Witold Dzwiniel, Jacek Kitowski, J. Moscinski and D. Yuen (Poland)

Posters

- *Co-Design Decisions for High Performance Parallel Architectures* 639
J.C. Moreno and A. Alcolea (Spain)
- *Achieving Data Availability on Parallel and Distributed File Systems* 645
Francisco Rosales and Raimundo Vega (Spain)
- *PC and DSP based AC motor adaptive vector control system* 651
David Juan Bedford Gaus, Antoni Arias Pujol, Emiliano Aldabas Rubira and José Luis Romeral Martínez (Spain)
- *Parallel Optimisation for Optical Lens Design* 657
Enric Fontdecaba Baig, José M. Cela Espín and Juan C. Dürsteler Lopez (Spain)
- *Supercomputer Optimised Microwave Domestic Oven Design via FD-TD* 663
Gaetano Bellanca, Paolo Bassi, Giovanni Erbacci, Gianni de Fabritiis and Ruggero Roccari (Italy)

- *Debugging Message Passing Parallel Applications: a General Tool* 669
Ana Paula Cláudio, João Duarte Cunha and Maria Beatriz Carmo (Portugal)
- *Parallel Ensemble-Averaged Molecular Dynamics Simulation of Shock Wave on Distributed Memory Multicomputers* 675
Sergey V. Zybin (Russia)
- *The Influence of Communication Patterns in the h-Relation Hypothesis in the IBM SP2* 681
J.L. Roda, C. Rodriguez, F. Almeida, D.G. Morales (Tenerife, Spain)
- *One-sided block Jacobi methods for the Symmetric Eigenvalue Problem* 687
D. Giménez, J. Cuenca, R. M. Ralha and A. J. Viamonte (Spain)
- *Efficient sparse data distribution for the Conjugate Gradient on distributed shared memory systems* 693
D.E. Singh, F.F. Rivera and J.C. Cabaleiro (Spain)
- *Synchronized Parallel Algorithms on Red Black trees* 699
Xavier Messeguer and Borja Valles (Spain)
- *Parallelization of GIS algorithms based on data partitioning* 705
M. Luisa Córdoba Cabeza and Antonio Pérez Ambite (Spain)
- *Emulating a superscalar processor to teach pipeline and superscalar concepts* 711
Santiago Rodríguez de la Fuente, M. Isabel García Clemente, Rafael Méndez Cavanillas and José M. Pérez Villadeamigo (Spain)
- *A Parallel Genetic Algorithm for Solving the Partitioning Problem in Multi FPGA Systems* 717
J. I. Hidalgo, M. Prieto, J. Lanchares and F. Tirado (Spain)
- *Haskell#: A Functional Language with Explicit Parallelism* 723
R.M.F.Lima and R.D.Lins (Brazil)
- *Parallel and Distributed Algorithm in State Estimation of Power System Energy* 729
J. Beleza Carvalho and F. Maciel Barbosa (Portugal)
- *Parallel Block Two-Stage Preconditioners for the Conjugate Gradient Method* 735
M. Jesus Castel, Violeta Migallón and José Penadés (Spain)

Technical Session 15

- *Neural Classifiers Implemented in a Transputer Based Parallel Machine* 839
J. M. Seixas, A. R. Anjos, C. B. Prado, L. P. Calôba, A. C. H. Dantas and J. C. R. Aguiar (Brazil)
- *Algorithm-Dependant Method to Determine the Optimal Number of Computers in Parallel Virtual Machines* 851
J.G. Barbosa and A.J. Padilha (Portugal)

Technical Session 16

- *Behaviour Analysis Methodology oriented to Configuration of Parallel, Real-Time and Embedded Systems* 865
F.J. Suárez, D.F. García (Spain)
- *Epsilon Balanced Decomposition for Power System Simulation on Parallel Computers* N.A.
Felipe Morales S. Hugh Rudnick V. D. W. Aldo Cipriano Z. (Chile)

Invited Talk 5

- *High Performance Computing for Image Synthesis* 879
Thierry Priol (France)

Technical Session 17

- *Modeling Snow Transport by Wind. A Cellular Automata* 895
Alexandre Masselot and Bastien Chopard (Switzerland)
- *Some Concepts of the software package FEAST* 907
Christian Becker, Susanne Kilian, Stefan Turek and John Wallis (Germany)
- *Dynamic Routing Balancing in Parallel Computer Interconnection Networks* 921
D. Franco, I. Garcés, E. Luque (Spain)

Technical Session 18

- *Calculation of Lambda Modes of a Nuclear Reactor: a Parallel Implementation using the Implicitly Restarted Arnoldi Method* 935
Vicente Hernández, José E. Román, Antonio M. Vidal, Vicent Vidal (Spain)

- *Parallelization of a Direct Method for Systems of Linear Equations* 741
M.F. Costa and R.M. Ralha (Portugal)

PART III

Technical Session 13

- *Parallel Genetic Algorithms for Hypercube Machines* 749
R. Baraglia and R. Perego (Italy)
- *Parallel Quadric Rendering with Load Balancing Strategy* 763
Dana Petcu (Romania)
- *Efficient Parallelization Approaches for the SAI Representation* 777
A. Sanchez, S. Campos and A. Rodriguez (Spain)
- *Parallel Implementations of Morphological Connected Operators Based on Irregular Data Structures* 791
Christophe Laurent and Jean Roman (France)

Technical Session 14

- *Dynamic Load Balancing in Crashworthiness Simulation* 805
H.G. Galbas and O. Kolp (Germany)
- *A Parallelization Strategy for Power Systems Composite Reliability Evaluation* 813
Carmen L.T. Borges and Djalma M. Falcão (Brazil)
- *Parallel Paradigms applied in a Fluid-Dynamic Problem to model a Glass Manufacturing Process* 825
J. Vinuesa, R. Menéndez de Llano, V. Puente and B. Torón (Spain)

- *Stochastic Control of the Scalable High Performance Distributed Computations* 949
Zdzislaw Onderka (Poland)
- *Direct Linear Solver for Vector and Parallel Computers* 963
Friedrich Grund (Germany)

Invited Talk 6

- *The Design of an ODMG Compatible Parallel Object Database Server* 977
Paul Watson (UK)

Technical Session 19

- *Parallel Query Processing in a Shared-Nothing Object Database Server* 1007
L.A.V.C. Meyer M.L.Q. Mattoso (Brazil)
- *High Performance Computing of a New Numerical Algorithm for an Industrial Problem in Tribology* 1021
M. Arenaz, R. Doallo, G. García and C. Vázquez (Spain)
- *Distributed Simulation Strategies of Graphite Electrode Forming Process* 1035
M. Danielewski, B. Bozek, K. Holly, G. Mysliwiec, J. Sipowicz and R. Schaefer (Poland)

Technical Session 20

- *Experimental Analysis of a Parallel Quicksort-Based Algorithm for Suffix Array Generation* 1049
Autran Macêdo, Elaine Spinola Silva, Denilson Moura Barbosa, Marco Antônio Cristo, João Paulo Kitajima, Berthier Ribeiro, Gonzalo Navarro and Nivio Ziviani (Brazil)
- *A Low Cost Distributed System for FEM Parallel Structural Analysis* 1063
C.O. Moretti, T.N. Bittencourt and L.F. Martha (Brazil)
- *Low Cost Parallelizing, a Way to be Efficient* 1077
Marc Martin and Bastien Chopard (Switzerland)

Parallel Genetic Algorithms for Hypercube Machines

Ranieri Baraglia and Raffaele Perego

Istituto CNUCE, Consiglio Nazionale delle Ricerche (CNR)
via S. Maria 36, Pisa, Italy
e-mail: {r.baraglia, r.perego}@cnuce.cnr.it

Abstract In this paper we investigate the design of highly parallel Genetic Algorithms. The Traveling Salesman Problem is used as a case study to evaluate and compare different implementations. To fix the various parameters of Genetic Algorithms to the case study considered, the Holland sequential Genetic Algorithm, which adopts different population replacement methods and crossover operators, has been implemented and tested. Both *fine-grained* and *coarse-grained* parallel GAs which adopt the selected genetic operators have been designed and implemented on a 128-node nCUBE 2 multicomputer. The *fine-grained* algorithm uses an innovative *mapping* strategy that makes the number of solutions managed independent of the number of processing nodes used. Complete performance results showing the behavior of Parallel Genetic Algorithms for different population sizes, number of processors used, migration strategies are reported.

1 Introduction

Genetic Algorithms (GAs) [11, 12] are stochastic optimization heuristics in which searches in solution space are carried out by imitating the population genetics stated in Darwin's theory of evolution. Selection, crossover and mutation operators, directly derived by from natural evolution mechanisms are applied to a population of solutions, thus favoring the birth and survival of the best solutions. GAs have been successfully applied to many NP-hard combinatorial optimization problems [6], in several application fields such as business, engineering, and science.

In order to apply GAs to a problem, a genetic representation of each individual (*chromosome*) that constitutes a solution of the problem has to be found. Then, we need to create an initial population, to define a cost function to measure the *fitness* of each solution, and to design the genetic operators that will allow us to produce a new population of solutions from a previous one. By iteratively applying the genetic operators to the current population, the fitness of the best individuals in the population converges to local optima.

Figure 1 reports the pseudo-code of the Holland genetic algorithm. After randomly generating the initial population $\beta(0)$, the algorithm at each iteration of the outer **repeat--until** loop generates a new population $\beta(t+1)$ from $\beta(t)$ by

selecting the best individuals of $\beta(t)$ (function `SELECT()`) and probabilistically applying the *crossover* and *mutation* genetic operators. The selection mechanism must ensure that the greater the fitness of an individual A_k is, the higher the probability of A_k being selected for reproduction. Once A_k has been selected, P_C is its probability of generating a son by applying the crossover operator to A_k and another individual A_i , while P_M and P_I are the probabilities of applying respectively, mutation and inversion operators to the generated individual respectively.

The crossover operator randomly selects parts of the parents' *chromosomes* and combines them to breed a new individual. The mutation operator randomly changes the value of a gene (a single bit if the binary representation scheme is used) within the chromosome of the individual to which it is applied. It is used to change the current solutions in order to avoid the convergence of the solutions to "bad" local optima.

The new individual is then inserted into population $\beta(t+1)$. Two main replacement methods can be used for this purpose. By adopting the *discrete* population model, the whole population $\beta(t)$ is replaced by new generated individuals at the end of the outer loop iteration. A variation on this model was proposed in [13] by using a parameter that controls the percentage of the population replaced at each generation. The *continuous* population model states, on the other hand, that the new individuals are soon inserted into the current population to replace older individuals with worse fitness. This replacement method allows potentially good individuals to be exploited as soon as they become available.

Irrespective of the replacement policy adopted, population $\beta(t+1)$ is expected to contain a greater number of individuals with good fitness than population $\beta(t)$. The GA end condition can be to reach a maximum number of generated populations, after which the algorithm is forced to stop or the algorithm converges to stable average fitness values.

The following are some important properties of GAs:

- they do not deal directly with problem solutions but with their genetic representation thus making GA implementation independent from the problem in question;
- they do not treat individuals but rather populations, thus increasing the probability of finding good solutions;
- they use probabilistic methods to generate new populations of solutions, thus avoiding being trapped in "bad" local optima.

On the other hand, GAs do not guarantee that global optima will be reached and their effectiveness very much depends on many parameters whose fixing may depend on the problem considered. The size of the population is particularly important. The larger the population is, the greater the possibility of reaching the optimal solution. Increasing the population clearly results in a large increase in GA computational cost which, as we will see later, can be mitigated by exploiting parallelism.

The rest of the paper is organized as follows: Section 2 briefly describes the computational models proposed to design parallel GAs; Section 3 introduces

```

Program Holland_Genetic_Algorithm;
begin
  t=0;
   $\beta(t) = \text{INITIAL\_POPULATION}()$  ;
  repeat
    for i = 1 to number_of_individuals do
       $F(A_i) = \text{COMPUTE\_FITNESS}(A_i)$ ;
       $\text{Average\_fitness} = \text{COMPUTE\_AVERAGE\_FITNESS}(F)$ ;
      for k = 1 to number_of_individuals do
        begin
           $A_k = \text{SELECT}(\beta(t))$ ;
          if ( $P_C > \text{random}(0,1)$ ) then
            begin
               $A_i = \text{SELECT}(\beta(t))$ ;
               $A_{child} = \text{CROSSOVER}(A_i, A_k)$ ;
              if ( $P_M > \text{random}(0,1)$ ) then  $\text{MUTATION}(A_{child})$ ;
               $\beta(t+1) = \text{UPDATE\_POPULATION}(A_{child})$ ;
            end
          end
        end;
    t=t+1;
  until (end_condition);
end

```

Figure1. Pseudo-code of the Holland Genetic Algorithm.

the Traveling Salesman Problem used as our case study, discusses the implementation issues and presents the results achieved on a 128-node hypercube multicomputer; finally Section 4 outlines the conclusions.

2 Parallel Genetic Algorithms

The availability of ever faster parallel computers means that parallel GAs can be exploited to reduce execution times and improve the quality of the solutions reached by increasing the sizes of populations managed.

In [5, 3] the parallelization models adopted to implement GAs are classified. The models described are:

- **centralized model.** A single unstructured *panmitic* population is processed in parallel. A master processor manages the population and the selection strategy and requests a set of slave processors to compute the fitness function and other genetic operators on the chosen individuals. The model scales poorly and explores the solution space like a sequential algorithm which uses the same genetic operators. Several implementations of centralized parallel GAs are described in [1].

- **fine-grained model.** This model operates on a single structured population by exploiting the concepts of *spatiality* and *neighborhood*. The first concept defines that a very small sub-population, ideally just an individual, is stored in one element (node) of the logical connection topology used, while the second specifies that the selection and crossover operators are applied only between individuals located on nearest-neighbor nodes. The neighbors of an individual determine all its possible partners, but since the neighbor sets of partner nodes overlap, this provides a way to spread good solutions across the entire population. Because of its scalable communication pattern, this model is particularly suited for massively parallel implementations. Implementations of fine-grained parallel GAs applied to different application problems can be found in [8, 9, 14, 15, 19].
- **coarse-grained model.** The whole population is partitioned into sub-populations, called *islands*, which evolve in parallel. Each island is assigned to a different processor and the evolution process takes place only among individuals belonging to the same island. This feature means that a greater genetic diversity can be maintained with respect to the exploitation of a panmitic population, thus improving the solution space exploration. Moreover, in order to improve the sub-population genotypes, a migration operator that periodically exchanges the best solutions among different islands is provided. Depending on the migration operator chosen we can distinguish between *island* and *stepping stone* implementations. In *island* implementations the migration occurs among every island, while in *stepping stone* implementations the migration occurs only between neighboring islands. Studies have shown that there are two critical factors [10]: the number of solutions migrated each time and the interval time between two consecutive migrations. A large number of migrants leads to the behavior of the island model similar to the behavior of a panmitic model. A few migrants prevent the GA from mixing the genotypes, and thus reduce the possibility to bypass the local optimum value inside the islands. Implementations of coarse grained parallel GAs can be found in [10, 20, 21, 4, 18, 16].

3 Designing parallel GAs

We implemented both *fine-grained* and *coarse-grained* parallel GAs applied to the classic Traveling Salesman Problem on a 128-node nCUBE 2 hypercube. Their performance was measured by varying the type and value of some genetic operators. In the following subsection the TSP case study is described and the parallel GA implementations are discussed and evaluated.

3.1 The Traveling Salesman Problem

The Traveling Salesman Problem (TSP) may be formally defined as follow: let $C = \{c_1, c_2, \dots, c_n\}$ be a set of n cities and $\forall i, \forall j$ $d(c_i, c_j)$ the distance between city c_i and c_j with $d(c_i, c_j) = d(c_j, c_i)$. Solving the TSP entails finding a permutation π' of the cities $(c_{\pi'(1)}, c_{\pi'(2)}, \dots, c_{\pi'(n)})$, such that

$$\sum_{i=1}^n d(c_{\pi'(i)}, c_{\pi'(i+1)}) \leq \sum_{i=1}^n d(c_{\pi^k(i)}, c_{\pi^k(i+1)}) \quad \forall \pi^k \neq \pi', (n+1) \equiv 1 \quad (1)$$

According to the TSP *path representation* described in [9], tours are represented by ordered sequences of integer numbers of length n , where sequence $(\pi(1), \pi(2), \dots, \pi(n))$ represents a tour joining, in the order, cities $c_{\pi(1)}, c_{\pi(2)}, \dots, c_{\pi(n)}$. The search space for the TSP is therefore the set of all permutations of n cities. The optimal solution is a permutation which yields the minimum cost of the tour.

The TSP instances used in the tests are: **GR48**, a 48-city problem that has an optimal solution equal to 5046, and **LIN105**, a 105-city problem that has a 14379 optimal solution¹.

3.2 Fixing the genetic operators

In order to study the sensitivity of the GAs for the TSP to the setting of the genetic operators, we used Holland's sequential GA by adopting the *discrete generation model*, one and two point crossover operators, and three different population replacement criteria.

- The *discrete generation model* separates sons' population from parents' population. Once all the sons' population has been generated, it is merged with the parents' population according to the replacement criteria adopted [8].
- One point crossover breaks the parents' tours into two parts and recombines them in the son in a way that ensures tour legality [2]. Two points crossover [7] works like the one point version but breaks the parents' tours into three different parts. A mutation operator which simply exchanges the order of two cities of the tour has been also implemented and used [9].
- The replacement criterion specifies a rule for merging current and new populations. We tested three different replacement criteria, called R1, R2 and R3. R1 replaces solutions with lower fitnesses of the current population with all the son solutions unaware of their fitness. R2 orders the sons by fitness, and replaces an individual i of the current population with son j only if the fitness of i is lower than the fitness of j . R2 has a higher control on the population than R1, and allows only the best sons to enter the new population. R3 selects the parents with a lower than average fitness, and replaces them with the sons with above average fitnesses.

The tests for setting the genetic operators were carried out by using a 640 solution population, a 0.2 mutation parameter (to apply a mutation to 20% of the total population), 2000 generations for the 48-city TSP, and 3000 generations

¹ Both the TSP instances are available at: <ftp://elib.zib-berlin.de/pub/mp-test-data/tsp/tsplib.html>

for the 105-city TSP. Every test was run 32 times, starting from different random populations, to obtain an average behavior. From the results of the 32 tests we computed:

- the average solution: $AVG = \frac{\sum_{i=1}^{32} F_{E_i}}{32}$, where F_{E_i} is the best *fitness* obtained with run E_i ;
- the best solution: $BST = \min\{F_{E_i}, i = 1, \dots, 32\}$;
- the worst solution: $WST = \max\{F_{E_i}, i = 1, \dots, 32\}$.

These preliminary tests allow us to choose some of the most suitable genetic operators and parameters for the TSP. Figure 2 plots the average fitness obtained by varying the crossover type on the 48-city TSP problem. The crossover was applied to 40% of the population and the R2 replacement criterion was used. As can be seen, the two point crossover converges to better average solutions than the one point operator. The one point crossover initially exhibits a better behavior, but after 2000 generations, converges to solutions that have considerably higher costs. We obtained a similar behavior for the other replacement criteria and for the 105-city TSP.

Table 1 reports *AVG*, *BST* and *WST* results for the 48-city TSP obtained by varying both the population replacement criterion and the percentage of the population to which the two point crossover has been applied. On average, the crossover parameter values in the range 0.4 – 0.6 lead to better solutions, almost irrespective of the replacement criterion adopted. Figure 3 shows the behavior of the various replacement criteria for a 0.4 crossover value. The R2 and R3 replacement criteria resulted in a faster convergence than R1, and they converged to very near fitnesses.

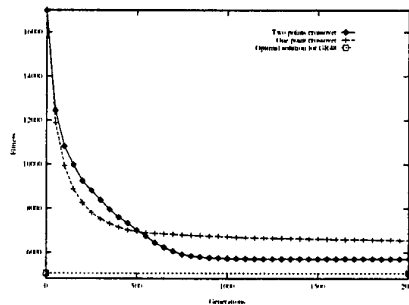


Figure2. Fitness values obtained with the execution of the sequential GA on the 48-city TSP by varying the crossover operator, and by using the R2 replacement criteria.

		Crossover parameter			
		0.2	0.4	0.6	0.8
R1	AVG	6255	5632	5585	5870
	BST	5510	5315	5135	5305
	WST	7828	6079	6231	6693
R2	AVG	5902	5696	5735	5743
	BST	5405	5243	5323	5410
	WST	7122	6180	6225	6243
R3	AVG	6251	5669	5722	5773
	BST	5441	5178	5281	5200
	WST	7354	6140	6370	6594

Table1. Fitness values obtained with the execution of the sequential GA on the 48-city TSP by varying the value of the crossover parameter and the population replacement criterion.

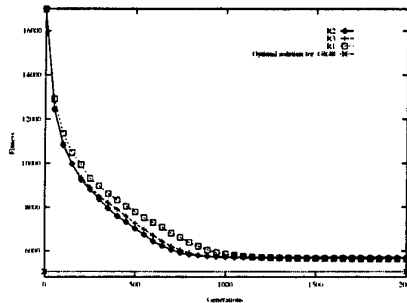


Figure3. Fitness values obtained by executing the sequential GA on the 48-city TSP with a 0.4 crossover parameter, and by varying the replacement criterion.

3.3 The coarse grained implementation

The coarse grained parallel GA was designed according to the *discrete generation* and *stepping stone* models. Therefore, the new solutions are merged with the current population at the end of each generation phase, and the migration of the best individuals among sub-population is performed among ring-connected islands. Each of the P processors manages N/P individuals, with N population size (640 individuals in our case). The number of migrants is a fixed percentage of the sub-population. As in [4], migration occurs periodically in a regular time rhythm, after a fixed number of generations.

In order to include all the migrants in the current sub-populations, and to merge the sub-population with the locally generated solutions, R1 and R2 replacement criteria were used, respectively. Moreover, the two point crossover operator was adopted.

Table 2 reports some results obtained by running the coarse grained parallel GA on the 48-city TSP. M denotes the migration parameter. The same data for a migration parameter equal to 0.1 are plotted in Figure 4. It can be seen that AVG, BST and WST solutions get worse values by increasing the number of the nodes used. This depends on the constant population size used: with 4 nodes sub-populations of 160 solutions are exploited, while with 64 nodes the sub-populations only consists of 10 individuals. Decreasing the number of solutions that forms a sub-population worsens the search in the solution space; small sub-populations result in an insufficient exploration of the solution space. The influence of the number of migrants on the convergence is clear from Table 2. When the sub-populations are small, a higher value of the migration parameter may improve the quality of solutions through a better mix of the genetic material.

		Number of processing nodes					
		4	8	16	32	64	128
$M=0.1$	AVG	5780	5786	5933	6080	6383	6995
	BST	5438	5315	5521	5633	5880	6625
	WST	6250	6387	6516	6648	8177	8175
$M=0.3$	AVG	5807	5877	5969	6039	6383	6623
	BST	5194	5258	5467	5470	5727	6198
	WST	6288	6644	7030	6540	8250	7915
$M=0.5$	AVG	5900	5866	5870	6067	6329	6617
	BST	5419	5475	5483	5372	6017	6108
	WST	6335	6550	7029	6540	8250	7615

Table2. Fitness values obtained with the execution of the coarse grained GA on the 48-city TSP by varying the migration parameter.

3.4 The fine grained implementation

The fine grained parallel GA was designed according to the *continuous generation model*, which is much more suited for fine grained parallel GAs than the *discrete* one. The two point crossover operator was applied.

According to the fine grained model the population is structured in a logic topology which fixes the rules of interaction between the solution and other solutions: each solution s is placed at a vertex $v(s)$ of logic topology T . The crossover operation can only be applied among nearest neighbor solutions placed on the vertices directly connected in T . Our implementation exploits the physical topology of the target multicomputer, therefore the population of 2^N individuals is structured as a N -dimensional hypercube.

By exploiting the recursivity of the hypercube topology definition, we made the number of solutions treated independent of the number of nodes used to execute the algorithm. As can be seen in Figure 5, a $2^3 = 8$ solution population

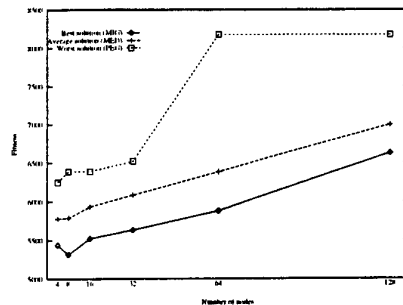


Figure 4. AVG, BST and WST values obtained by executing the coarse grained GA on the 48-city TSP as a function of the number of nodes used, and 0.1 as migration parameter.

can be placed on a $2^2 = 4$ node hypercube, using a simple mapping function which masks the first (or the last) *bit* of the **Grey** code used to numerate the logical hypercube vertices [17]. Physical node X00 will hold solutions 000 and 100 of the logic topology, not violating the neighborhood relationships fixed by the population structure. In fact, the solutions on the neighborhood of each solution s will still be in the physical topology on directly connected nodes or on the node holding s itself.

We can generalize this *mapping* scheme: to determine the allocation of a 2^N solution population on a 2^M node physical hypercube, with $M < N$, we simply mask the first (the last) $N - M$ *bits* of the binary coding of each solution.

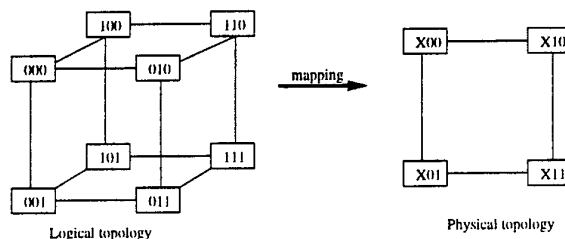


Figure 5. Example of an application of the *mapping* scheme.

Table 3 reports the fitness values obtained with the execution of the fine grained GA by varying the population dimension from 128 solutions (a 7 dimension hypercube) to 1024 solutions (a 10 dimension hypercube). As expected, the ability to exploit population sizes larger than the number of processors used in our mapping scheme, leads to better quality solutions especially when few pro-

cessing nodes are used. The improvement in the fitness values by increasing the number of nodes while maintaining the population size fixed, is due to a particular feature of the implementation, which aims to minimize the communication times to the detriment of the diversity of the same node solutions. Selection rules tend to choose partner solutions in the same node. The consequence is a greater uniformity in solutions obtained on few nodes, which worsens the exploration of the solution space. The coarse grained implementation suffered of the opposite problem which resulted in worse solutions obtained as the number of nodes was increased. This behavior can be observed by comparing Figure 4, concerning the coarse grained GA, and Figure 6, concerning the fine grained algorithm with a 128 solution population applied to the 48-city TSP.

Table 4 shows that an increase in the number of solutions processed results in a corresponding increase in the speedup values. This is because a larger number of individuals assigned to the same processor leads to lower communication overheads for managing the interaction of each individual with neighbor partners.

Population size		Number of processing nodes						
		1	4	8	16	32	64	128
128	AVG	39894	24361	23271	23570	23963	22519	22567
	BST	34207	20774	19830	20532	21230	20269	20593
	WST	42127	30312	26677	27610	27634	28256	25931
256	AVG	33375	25313	22146	21616	21695	22247	21187
	BST	29002	24059	20710	19833	20144	19660	19759
	WST	40989	26998	23980	24007	23973	24337	22256
512	AVG	28422	23193	22032	21553	20677	20111	20364
	BST	28987	22126	19336	20333	19093	18985	18917
	WST	41020	25684	23450	22807	22213	21696	21647
1024	AVG	25932	23659	22256	20366	19370	18948	19152
	BST	27010	21581	21480	18830	18256	18252	17446
	WST	40901	25307	22757	21714	20766	19525	20661

Table3. Fitness values obtained with the execution of the fine grained GA applied to the 105-city TSP after 3000 generations, by varying the number of solutions per node.

3.5 Comparisons

We compared the fine and coarse grained algorithms on the basis of the execution time required and the fitness values obtained by each one after the evaluation of 512000 solutions. This comparison criterion was chosen because it allows to overcome computational models diversity that make non comparable the fine and coarse grained algorithms. The evaluation of 512000 new solutions allows both the algorithms to converge and requires comparable execution times. Table 5 shows the results of this comparison. It can be seen that when the number of

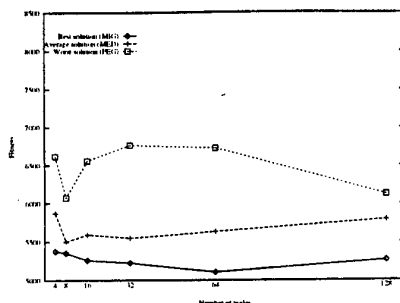


Figure6. AVG, BST and WST values obtained by executing the fine grained GA applied to the 48-city TSP as a function of the number of nodes used. The population size was fixed to 128 individuals.

Number of nodes	Number of individuals			
	128	256	512	1024
1	1	1	1	1
4	3.79	3.84	3.89	3.92
8	7.25	7.47	7.61	7.74
16	13.7	14.31	14.78	15.12
32	25.25	27.02	28.03	29.38
64	44.84	49.57	53.29	56.13
128	78.47	88.5	98.06	105.32

Table4. Speedup of the of the fine grained GA applied on the 105-city TSP for different population sizes.

the nodes used increases the fine grained algorithm gets sensibly better results than the coarse grained one. On the other hand, the coarse grained algorithm shows a super-linear speedup due to the *quick sort* algorithm used by each node for ordering by fitness the solutions managed. As the number of nodes is increased, the number of individuals assigned to each node decreases, thus requiring considerably less time to sort the sub-population.

4 Conclusions

We have discussed the results of the application of parallel GA algorithms to the TSP. In order to analyze the behavior of different replacement criteria and crossover operators and values Holland's sequential GA was implemented. The tests showed that the two point crossover finds better solutions, as does a replacement criteria which replaces an individual i of the current population with son j only if the fitness of i is worse than the fitness of j . To implement the *fine-grained* and *coarse-grained* parallel GAs on a hypercube parallel com-

	Number of processing nodes					
	4	8	16	32	64	128
Fine grained						
AVG	26373	26349	25922	25992	24613	23227
BST	23979	23906	23173	21992	22145	20669
WST	30140	27851	29237	29193	30176	26802
Execution times	1160	606	321	174	98	51
Coarse grained						
AVG	23860	24526	27063	29422	32542	35342
BST	20219	21120	23510	25783	30927	33015
WST	25299	29348	36795	39330	39131	41508
Execution times	1670	804	392	196	97	47

Table5. Fitness values and execution times (in seconds) obtained by executing the fine and coarse grained GA applied to the 105-city TSP with a population of 128 and 640 individuals, respectively.

puter the most suitable operators were adopted. For the coarse grained GA we observed that the quality of solutions gets worse if the number of nodes used is increased. Moreover, due to the sorting algorithm used to order each sub-population by fitness, the speedup of the coarse grained GA were super-linear. Our fine-grained algorithm adopts a mapping strategy that allows the number of solutions to be independent of the number of nodes used. The ability to exploit population sizes larger than the number of processors used gives better quality solutions especially when only a few processing nodes are used. Moreover, the quality of solutions does not get worse if the number of the nodes used is increased. The fine grained algorithm showed good scalability. A comparison between the fine and coarse grained algorithms highlighted that fine grained algorithms represent the better compromise between quality of the solution reached and the execution time spent on finding it.

The GAs implemented reached only "good" solutions. In order to improve the quality of solutions obtained, we are working to include a local search procedure within the GA.

References

1. R. Bianchini and C.M. Brown. Parallel genetic algorithm on distributed-memory architectures. Technical Report TR 436, Computer Sciences Department University of Rochester, 1993.
2. H. Braun. On solving travelling salesman problems by genetic algorithms. In *Parallel Problem Solving from Nature - Proceedings of 1st Workshop PPSN*, volume 496 of *Lecture Notes in Computer Science*, pages 129-133. Springer-Verlag, 1991.
3. E. Cantu-Paz. A summary of research on parallel genetic algorithms. Technical Report 95007, University of Illinois at Urbana-Champaign, Genetic Algorithms Lab. (IlligAL), <http://gal4.ge.uiuc.edu/illigal.home.html>, July 1995.

4. S. Cohoon, J. Hedge, S. Martin, and D. Richards. Punctuated equilibria: a parallel genetic algorithm. *IEEE Transaction on CAD*, 10(4):483-491, April 1991.
5. M. Dorigo and V. Maniezzo. Parallel genetic algorithms: Introduction and overview of current research. In *Parallel Genetic Algorithms*, pages 5-42. IOS Press, 1993.
6. M. R. Garey and D.S. Jonshon. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.
7. D. Goldberg and R. Lingle. Alleles, loci, and the tsp. In *Proc. of the First International Conference on Genetic Algorithms*, pages 154-159, 1985.
8. M. Gorges-Schleuter. Explicit parallelism of genetic algorithms through population structures. In *Parallel Problem Solving from Nature - Proceedings of 1st Workshop PPSN*, volume 496, pages 398-406. Lecture Notes in Computer Science, 1990.
9. M. Gorges-Schleuter. *Genetic Algorithms and Population Structure*. PhD thesis, University of Dortmund, 1991.
10. P. Grosso. *Computer Simulations of Genetic Adaptation: Parallel Subcomponent Interaction in a Multilocus Model*. PhD thesis, University of Michigan, 1985.
11. J. Holland. *Adaptation in Natural and Artificial Systems*. Univ. of Michigan Press, 1975.
12. J.H. Holland. Algoritmi genetici. *Le Scienze*, 289:50-57, 1992.
13. K.A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975.
14. B. Manderick and P. Spiessens. Fine-grained parallel genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 428-433. Morgan Kaufmann Publishers, 1989.
15. H. Muhlenbein. Parallel genetic algorithms, population genetic and combinatorial optimization. In *Parallel Problem Solving from Nature - Proceedings of 1st Workshop PPSN*, volume 496, pages 407-417. Lecture Notes in Computer Science, 1991.
16. H. Muhlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. *Parallel Computing*, 17:619-632, 1991.
17. nCUBE Corporation. ncube2 processor manual. 1990.
18. C. Pettey, M. Lenze, and J. Grefenstette. A parallel genetic algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 155-161. L. Erlbaum Associates, 1987.
19. M. Schwehm. Implementation of genetic algorithms on various interconnections networks. *Parallel Computing and Transputer applications*, pages 195-203, 1992.
20. R. Tanese. Parallel genetic algorithms for a hypercube. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 177-183. L. Erlbaum Associates, 1987.
21. R. Tanese. Distributed genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 434-440. M. Kaufmann, 1989.

Parallel Quadric Rendering with Load Balancing Strategy

Dana Petcu

West University of Timișoara, B-dul V.Pârvan 4, 1900 Timișoara, Romania,
petcu@info.uvt.ro,

WWW home page: <http://www.info.uvt.ro/~petcu>

Abstract. The paper presents a parallel algorithm for generating conics on a raster scan display device. Unlike the Wright's algorithms [11] for drawing lines and circles, which are based on geometric decompositions of the curves, the new algorithm leads to an equal distribution of continuous sets of pixels between the drawing processes. As a practical application, a parallel z-buffer algorithm for quadric surfaces was introduced. The classical large set of filled polygons was replaced by a set of conics representing some section curves of the quadrics. Tests have been performed on a transputer machine and on a distributed network¹.

1 Introduction

Parallel rendering offers the potential for high performance. Realizing this potential requires a careful analysis of all the algorithms involved, especially in the generation of life-like images, known as realistic image synthesis, since the techniques falling in this category are notorious for their high computational complexity. A such analysis must start, first of all, with the drawing algorithms of the basic graphics primitives. Three major performance bottlenecks consistently resist attempts to increase rendering speed: the number of floating-point operations to perform geometrical calculations, the number of integer operations to compute pixel values, and the number of frame-buffer accesses to store the image and to determine visible surfaces. Several issues must be considered while designing a parallel system. Some of the more important of them are load balancing, network congestion, parallelization overheads, coherency exploitation, algorithm embedding, and suitability to general-purpose parallel computers. Although parallelism has been employed in computer graphics since the early days of the field, applying it effectively is a complex problem.

1.1 Parallel scan-conversion algorithms for conics

Scan-conversion algorithms use incremental methods to minimize the number of calculations performed during each iteration. Bresenham's classic algorithm [1] for drawing lines and circles is attractive because it uses only integer arithmetic.

¹ This work was partially supported by IWR Institute, University of Heidelberg

For the more general case of a conic, special incremental algorithms, like those presented in [4] or [3], have been constructed using Bresenham's idea.

The Wright's papers [11] and [10] report on some methods of achieving parallelism in a raster graphics system by paralleling the Bresenham's algorithms for drawing lines and circles. The parallel circle algorithm presented by Wright divides the work among p processors splitting the $\pi/4$ arc from north to north-east (the octant 2, the rest of the circle is constructed by symmetries) into p approximately equal subarcs (with the same length), and assigning a different processor to determine the raster representation for each subarcs. Wright's remark was that the times for different processors are not necessarily the same or similar, primarily due to the fact that the numbers of pixels treated by distinct processors are not equal. The same remark we can make in the case when we try to draw a large set of circles and we apply a pool of tasks technique, a task being the drawing procedure of a circle.

Rather than dividing the circular arc into many equal sub-arcs, the algorithm proposed recently by Huang and Banissi [6] segments the horizontal length of octant 2 into many equal parts. Some arrangement have been made to include only multiplication, addition and shift operations. Values of sine and cosine functions must be pretabulated outside the iteration loop.

1.2 Modeling systems based on quadrics

Most surface-rendering systems render a set of polygons that approximate the model representation. Polygons are simple, regular primitives that are convenient to display, so that polygon rendering is usually more efficient and numerically robust than direct surface rendering. Unfortunately, the polyhedral model is only an approximation to the real surface and frequently aliasing and inaccuracies can occur (especially for complex, curved surfaces). Other representations, such as section curves patches, are more convenient to specify when modeling a curved surface. Rendering the surface as a set of curves is appealing because the representation of each curve is exact (the need for the anti-aliasing techniques developed for rendering polygonal approximations is also eliminated).

Quadric surfaces are particularly useful in specialized applications such as molecular modeling, and have also been integrated into geometric and solid modeling systems, like that presented in [7]. The natural quadrics (the spheres, the circular cylinders and the right cones) are the most natural objects to model mechanical parts. The reasons for using quadrics include ease of computing the surface normal, testing whether a point is on the surface, computing the depth of the point which was projected into a plane (important in hidden-surface algorithms), calculating intersections of one surface with another [4].

1.3 Rendering curved surfaces

The image synthesis supposes three distinct phases [9]: *preprocessing* – consists in data read, transformation of points, clipping, projection; *rendering* – includes

hidden-surface removal, shading, and any other visual effects; *postprocessing* – involves displaying the image on a frame buffer.

Volume rendering is a computationally intensive task. Current graphics computers are not fast enough to generate high quality images at interactive speeds. High resolution volume rendering can take minutes to hours on a uniprocessor workstation and the rendering time usually grows linearly with the data size. Moreover, volume data sets can be very large, often too large for a workstation to hold in memory at once.

Surface rendering is traditionally performed by approximating the surface with polygons and then rendering the polygons. Almost all visible-surface algorithms were described for objects defined by polygonal facets (one exception in the z-buffer algorithm which does not require that objects be polygons). Objects such as the curved surfaces must first be approximated by many small facets before polygonal versions of any of these algorithms can be used. Although a such approximation can be done, it is often preferable to scan convert curved surfaces directly, eliminating polygonal artifacts and avoiding the extra storage required by polygonal approximation. Special visible-surface algorithms for quadrics have been developed (they all find the intersections of two quadrics, yielding a complicated equation whose roots must be found numerically [4]).

Recently several papers presented methods that render surfaces as sequences of curves [2]. These methods have deficiencies in their ability to guarantee a complete coverage of the render surface, in their ability to prevent processing the same pixel multiple times, or their ability to produce an optimal surface coverage.

One of the most powerful attractions of z-buffer algorithm is that it can be used to render any object if a z-value can be determined for each point in its projection. No explicit intersection algorithms need to be written.

1.4 Load balancing

The load balancing strategies associated with an image-space visible-surface algorithm, like z-buffer, can be divided with respect to how specific tasks are determined, into data nonadaptive and data adaptive [9].

The data nonadaptive methodology relies on an initial decomposition of image space unrelated to the input data. Many easily constructed image-space tasks of varying work loads are assigned (statically or dynamically) for parallel processing (dynamic assignment typically produces better load balancing compared to the static assignment). Typically the image is tiled into rectangular areas, and a work-queue approach is used. When a processor starts rendering, it is assigned a region and renders it. When it finishes the region, it asks for another region and continues this loop until the image is done. The larger is the number of areas, the more work is involved in preprocessing, communication and object duplication, but the better the load balancing.

In the data adaptive case, the sizes of the tasks (the area of the pixel regions) are adjusted according to the input data in an attempt to obtain better load balancing. Data adaptive schemes seems to be less suitable for generating single

images than data nonadaptive algorithms due to an additional preprocessing overhead.

1.5 Goals

We present in the next two sections a parallel algorithm for solving the problem of drawing a two-dimensional curve by distributing the computational effort equally to a moderate number of processors. In particular, we obtain a parallel algorithm (described also in [8]) for scan converting general conics (ellipses, circles, parabolas, hyperbolas) based on the Van Aken's sequential (incremental) algorithm described in [4].

In the third section we present an application of this algorithm for solving the problem of drawing a set of quadric surfaces. We have developed a parallel algorithm for quadric rendering, which distributes the image building and combining processes. Basically, the rendering is done using an algorithm of z-buffer type. Only two-dimensional partial images are communicated among processors and not three-dimensional volume data. The image combining operation is essentially the composition operation applied to some two-dimensional images in parallel. The algorithm guarantees a complete (dynamic) coverage of the render surface.

Load balancing is achieved distributing equal parts of each scene object to the available processors and render them independently from each other (a data adaptive strategy).

The proposed algorithm was implemented in two different environments: on a parallel machine, a T-800 multiprocessor, using PARIX system for C language (message passing parallel architecture), and on a distributed network, with four identical SUN stations connected on a network, using PVM 3.0 (Parallel Virtual Machine). The goal of the tests is twofold: (i) to investigate to what extent the theoretical parallelization can be realized in practice (on a parallel computer or on a distributed system); (ii) to compare the proposed algorithm with similar (sequential or parallel) algorithms in order to underline the efficiency of the new algorithm and the advantage of the proposed load balancing strategy.

2 Computing the iteration number of the incremental algorithm

Trying to generalize the Wright's idea in the case of a planar continuous curve, we see that the mathematical problem of dividing it into equal parts is not very simple. Moreover, mathematically equal (continuous curve) parts are not represented on the display by the same number of pixels. Therefore, the iteration numbers of the basic incremental algorithm (Bresenham's algorithm for circles, for example) differs from one curve part to another.

Note that we can easily solve the problem of finding the extreme points of the curve, especially when we treat a second degree curve.

We state that for a planar curve C generated by a function g twice differentiable and with continuous second derivatives, we can find the number of pixels of the discretized curve which represents, at a 1:1 scale, the continuous curve on raster display.

Suppose we describe the curve C implicitly by a function $g(x, y)$ and we draw the discretized curve with an incremental algorithm.

When the gradient vector has a slope $|m| > 1$ (in Figure 1.(a), for example, from (x_1, y_1) to (x_2, y_2) , or from (x_4, y_4) to (x_6, y_6)), from one incremental step to another the x value is incremented with ± 1 . When the gradient vector has a slope $|m| \leq 1$ (from (x_2, y_2) to (x_4, y_4) , for example) from one incremental step to another the y value is incremented with ± 1 . If we establish a direction of plotting the curve $g(x, y) = 0$, for example $g(x, y) < 0$ on the left and $g(x, y) > 0$ on the right, the sign of the increment can be properly chosen.

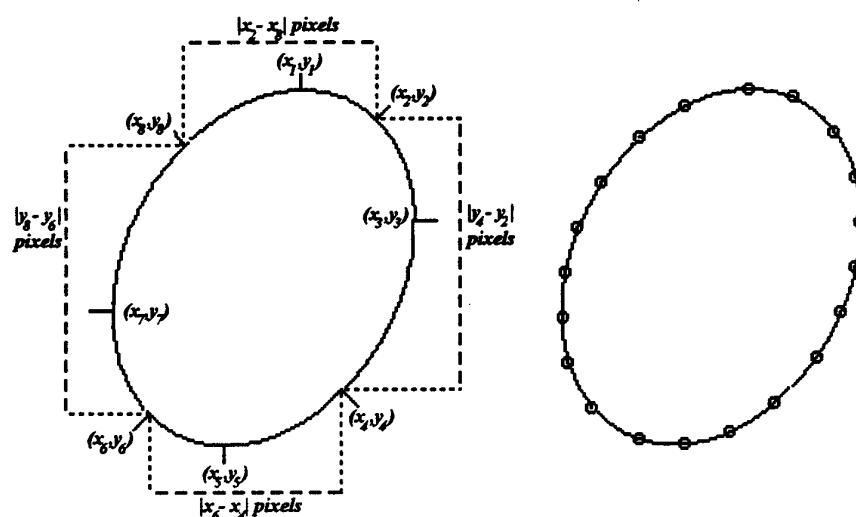


Fig. 1. A discretized (scan-converted) ellipse (a) Numbering its pixels (b) Dividing it into twenty parts with equal number of pixels

In the conic case, the Van Aken's incremental algorithm [4] (dominated by integer operations) can be used in the drawing process.

In order to find the pixel number of the discretized curve, we must find the points (x, y) where the gradient is 0 (extreme pixels), 1 or -1 (where a change in the incremented variable will be made). These points can be found by solving:

$$\frac{\partial g}{\partial x}(x, y) = 0, \quad \frac{\partial g}{\partial y}(x, y) = 0, \quad \left(\frac{\partial g}{\partial x} + \frac{\partial g}{\partial y} \right)(x, y) = 0, \quad \left(\frac{\partial g}{\partial x} - \frac{\partial g}{\partial y} \right)(x, y) = 0.$$

in the unknown (x, y) .

In the case of a second degree curve, these equations are not so complicated. There are, maximum, eight solutions (in Figure 1.(a) these solutions are denoted by (x_i, y_i) , $i = 1, \dots, 8$). If the curve is a hyperbola or a parabola, we must test which solution points are inside the screen domain, and we must perform the intersections with the edges of the screen rectangle.

In the case of an arbitrary function g , solving the above equations can be a difficult problem, and, therefore, the computational overhead, introduced by dividing the curve, can affect the execution time of a specific implementation of the algorithm.

Using the above remarks we can compute the pixel number of the discretized curve (for example, in the case of Figure 1.(a), the pixel numbers is $|x_2 - x_8| + |x_6 - x_4| + |y_4 - y_2| + |y_8 - y_6|$).

3 Parallel algorithm for drawing a conic

The main difficulty for each processor involved in a parallel incremental algorithm for drawing a basic graphic primitive is to jump into the middle of the calculations that are normally performed sequentially by a single processor.

Suppose we know a point on the discretized curve (the initial point in the sequential incremental algorithm), and we want to draw simultaneously all the p -parts of a curve. The starting and the ending pixels of a particular p -part of the discrete curve are computed as follows. The starting pixel is the ending pixel of the previous p -part of the curve with the exception of the first part of the curve for which we use the initial point as starting pixel. We establish the number of pixels of the current p -part of the curve (the same as for the previous p -part of the curve, with the exception of the last p -part). After that, we find the ending point: first, we get an x or y value using the above number and the number of pixels to be draw in each direction from the starting pixel; second, we get the other coordinate, y or x , solving the equation $g(x, y) = 0$. In Figure 1.(b) the starting and ending points are emphasized for the ellipse of Figure 1.(a) and $p = 20$.

After getting started, the logic for each processor is identical to the logic used in the sequential (incremental) algorithm except for a small change in the stopping criterion (we have arrive to the ending point or not).

Figure 2 shows how it can be obtained curves like conics or graphs of some functions using three drawing processes.

In practice, an implementation of the parallel algorithm on a distributed-memory architecture will be efficient only if the creation of an image part will be more expensive in time than the additional communications required to send the partial images and the initial data between the processes. Therefore, we expect that the efficiency of the parallel algorithm can be proved when we deal with a large set of curves.

Suppose we have a collection of quadrics (ellipsoids, spheres, cons, cylinders, paraboloids or hyperboloids) which must be represented in a two-dimensional image using a parallel or a perspective projection.

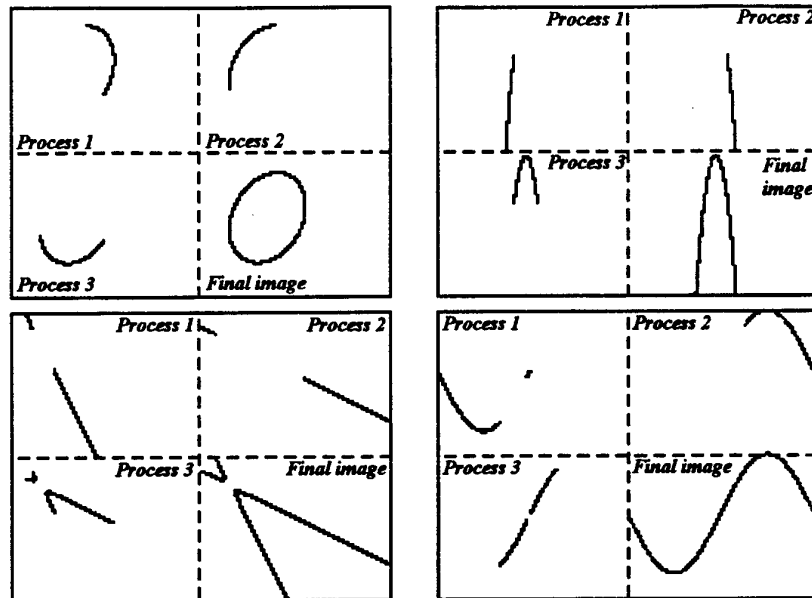


Fig. 2. Three processes are working to produce an image of (a) an ellipse (b) a parabola (c) an hyperbola (d) a sinusoidal function

Note that the intersection of the quadric with a plane is a conic. In the case of a parallel projection, we can use the parallel algorithm to represent a wire-frame model of the scene composed by these quadrics. An example of a such scene is given in Figure 3. If the intersection plane is parallel with the projection plane, the (parallel or perspective) projection of the intersection curve is also a conic. We can exploit this observation in a z-buffer like algorithm.

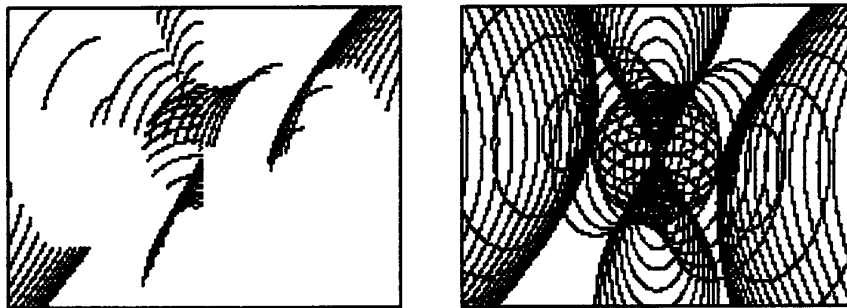


Fig. 3. Wire-frame model of a scene composed by some quadrics: (a) the 5th part of the image constructed by one processor (b) the final image

4 Rendering a quadric

Theoretically, we can generate an two-dimensional image of a quadric using the intersection curves of the quadric with a set of parallel planes with the projection plane.

The main problem is the distance between two consecutive intersection planes, i.e. how we can choose this distance thus that no pixels between two curves will appear, and the number of the overlapped pixels is minimized. The distance between consecutive planes varies in relation with the quadric form.

We start from a front-end plane. At a particular step, we can compute the optimal distance using the difference between the extreme pixels of the current intersection curve and those of the last intersection curve. Each difference between corresponding pairs must be no more than one, and at least one such difference must be nonzero. If this condition is not fulfilled another intersection plane must be selected (closer or farther from the last one).

Each processor of the parallel/distributed system with p processors is responsible for drawing the image of a p -part of each scene object; it draws effectively a p -part of each section curve. For any pixel of a section curve, a z -value can be recovered from the equation of the intersection plane. The extreme pixels of the intersection curves and the distances to the next section planes must be found by each processor (insignificant relative to the total computation time when the number of processors is small).

The Warn's lighting model [4] was used in our implementation of the rendering process.

Suppose that the input data for each processor are the set of coefficients of all quadrics. Then the local memory of each processor must have only the capacity to store a z -buffer, a local array of pixel colors, and the coefficients.

Theoretically, each z -buffer and each array of pixel colors must be sent to the processor which display the final image. In practice, it is more convenient to send only the z -values and the colors of the pixels which have been drawn (for example, the component pixels of the 3rd part of the sphere from Figure 4(a)).

Note that the proposed algorithm do not split the z -buffer between the processors (like the parallel z -buffer algorithms presented in [5] or [9]). Instead, each quadric is divided so that the number of pixels treated by each processor is the same. Note also that the data storing requirements for each processors are least that in the case of applying the z -buffer algorithm for a polyhedral approximation (in particular, in the case of Figure 5, at least 14000 polygons are necessary for obtaining a similar image).

We expect that the computational overhead introduced by the dividing procedure of each object into equal parts will have a maximum negative effect on the efficiency of the parallel algorithm if the patches curves will be very small (in this case, the incremental algorithm has a small number of iterations which must be distributed between the available processors). In order to analyze a such case we have constructed the model presented in Figure 6. As the tests will prove, this computational overhead is no more expensive in time than 15% of the rendering process.

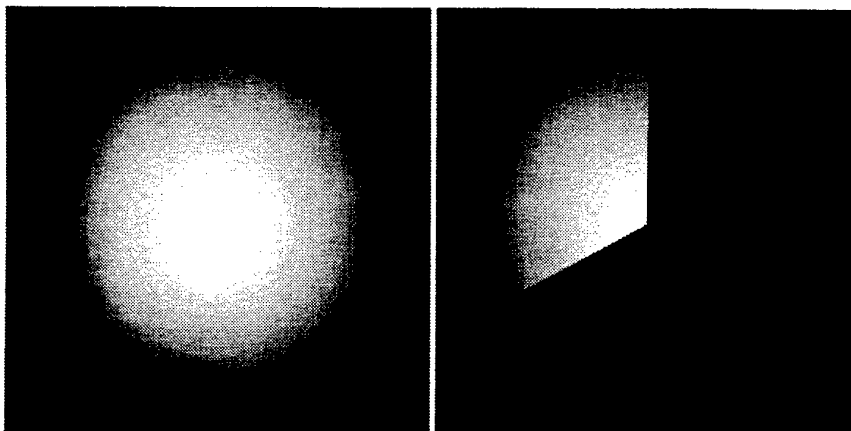


Fig. 4. Drawing a sphere: (a) the final image (b) the image produced by a processor in the case of a group of $p = 3$ processors

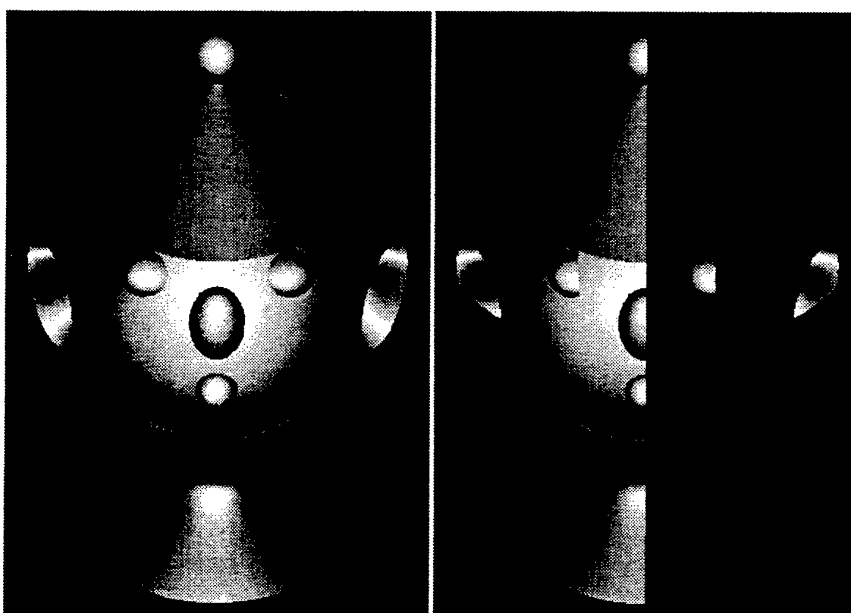


Fig. 5. Image composed of natural quadrics: (a) the final image (b) half-image created by one processor

The resulting data from all the processors must be combined to form the final image. Combining can be done algorithmically in several different ways. The

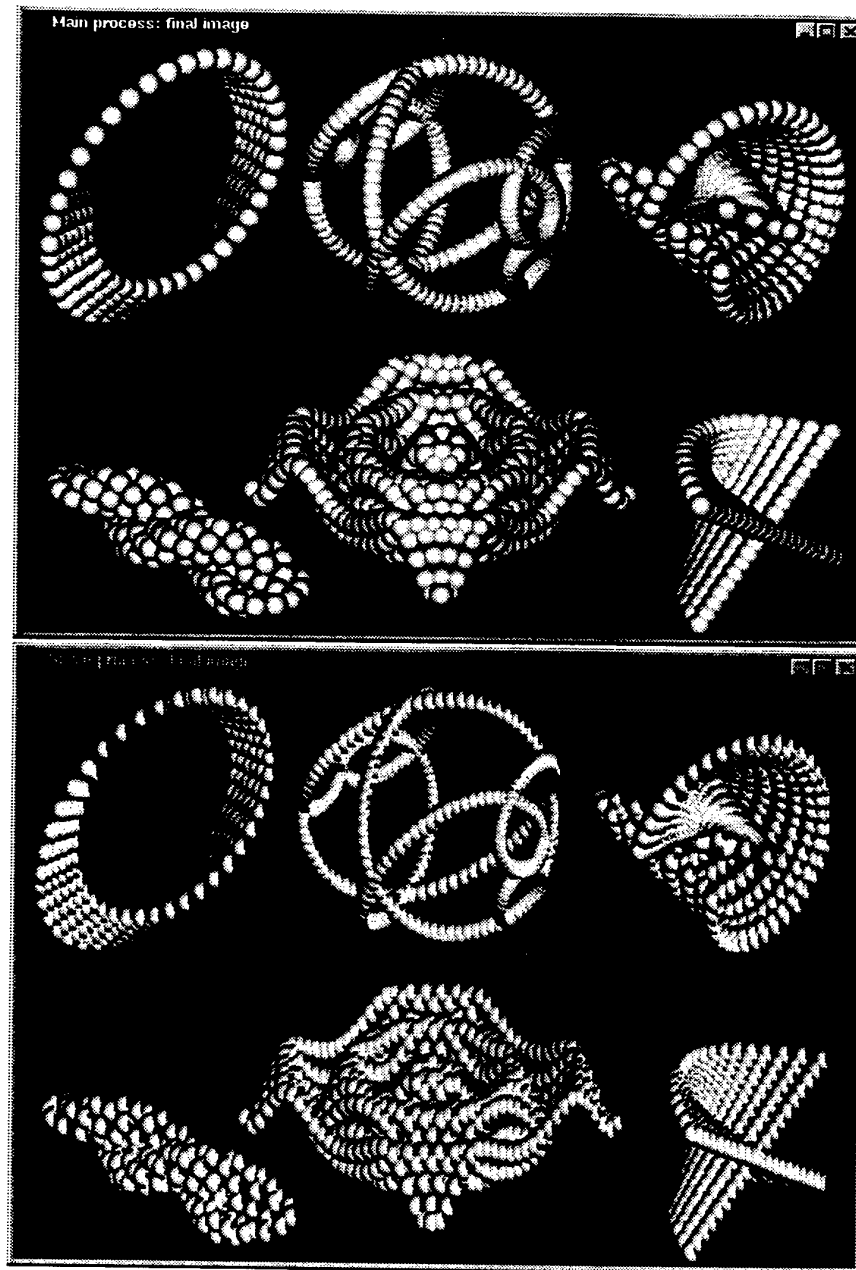


Fig. 6. Geometric models of some well-known parametrical surfaces using small spheres: (a) the final image (b) half-image created by one processor

choice of one method of combining over the other can only be determined by the size of the data, the number of processors, the topology of the inter-connection network and the routing strategy used. In a centralized scheme all processors send their local images to one processor. Although simple to implement, this strategy is inefficient in the presence of a large number of processors. The reason lies in the network congestion that arises when all processors attempt to send their data to a single distinguished processor. Some comparative studies [12] of the different methods of combining conclude that for fewer number of processors, a simple tree scheme is more suitable. Therefore, we have employ a strategy that take advantage of the inherent tree-like structure of composition (image combining belongs to a class of associative operations which includes, for example, addition). In $\log p$ steps the final image can be created.

5 Experimental results

We denote by T_0 the running time (in seconds) of the sequential implementation of the basic (Van Aken's) incremental algorithm (with one process on one processor), by T_1 the running time of the sequential implementation of the concurrent algorithm (with one process on one processor), and by T_p the running time of the parallel or distributed implementation of the parallel algorithm with p processes distributed to p processors. Then the efficiency of the concurrent algorithm implemented in a sequential mode compared with the classical incremental algorithm is given by the relationship $E_1^{comp} = T_0/T_1$, the efficiency of the implementation of the concurrent algorithm on p processors of a (simulated or real) parallel machine is given by the relationship $E_p^{par} = T_1/(pT_p)$, and the efficiency of the proposed algorithm using a (simulated or real) parallel machine is given by the relationship $E_p = T_0/(pT_p)$ (E_p^{Pari} on the parallel machine, and E_p^{Pvm} on the distributed network).

We have use a master-slave computing scheme. The master starts the slave processes and, if it is necessary, it send them some initial data. Each slave finishes the job and sends (using some intermediate processes according to a tree structure) the result back to the master which will display the final image. The algorithm running on each node processor consists of two main parts: the first is an outer body containing the counterparts of the message-passing calls, and the second is an inner loop for the rendering algorithm.

In order to design the communication procedures, we can adopt one of the following strategy: (1) the master computes the section curves and the associated depths, and send the equation of each section curve, the starting and the ending points of the i part of the curve to the i th processor, and each slave is occupied only with the render process; (2) the master computes the equations of the section curves and broadcast them with the associated depths, and each slave must find the starting and the ending points of the corresponding part of the curve, and renders it; (3) the master broadcasts the set of quadrics to be render and each slave computes the section curves, find its part of the curve, and renders

it; (4) each slave read/generates the set of quadrics to be render, generates the section curves, and renders the corresponding parts.

Each above mentioned strategy has been tested. For our test images presented in the last section, the first two strategy were too expensive in time, since the communication time has overpass the computing time (especially in the case of the implementation on a workstation network). Therefore, the following results refer to an implementation of the third strategy in the case of Figures 3, 4, 5, and of the fourth strategy in the case of Figure 6 (in this particular case, it is more easier to generate the sequence of quadrics than to send each quadric equation from master to slaves).

Experiment 1: Computational efficiency. We have study the influence on the running time due to the computational overhead for splitting the curves in equal parts. The efficiency of the concurrent algorithm implemented in a sequential way (compared with the classical incremental algorithm) is almost 1 if the discretized conics have a number of pixels comparable with the dimensions of the final image (see Table 1). Note that the last column corresponds to the worst case when the

Table 1. Computational efficiency of the algorithm applied to the models from Figs.1-6

No. curves	1	130	226	1307	15657	233993
E_1^{comp}	98%	95%	94%	99%	99%	86%

section curves are very small (Figure 6). If we have a more complicated curve (or a set of curves) than a second degree one (ones), E^{comp} decreases since it is possible that the partial differential equations are not linear and we must solve them with a Newton-type procedure.

Experiment 2: Efficiency of the parallel implementation. Comparing the running time of the code for the algorithm on a single one processor (without communication overhead) with those on more processor, we have obtained the results which are presented in Table 2.(a). As it can be seen, the parallel implementa-

Table 2. Parallel and distributed implementation efficiency: (a) E_p^{par} for different values of p (b) E_3 for different computer architecture

No. curves	E_2^{par}	E_3^{par}	E_4^{par}	E_5^{par}	E_6^{par}	E_7^{par}	E_8^{par}	E_3^{Pvm}	E_3^{Parix}
1	44.68%	29.79%	21.00%	14.00%	10.61%	8.45%	6.56%	12%	29%
130	87.02%	78.87%	71.82%	64.93%	54.85%	48.25%	43.51%	51%	78%
226	91.91%	85.29%	80.99%	75.99%	73.77%	67.81%	60.68%	67%	84%
401	89.93%	86.43%	81.28%	76.57%	75.25%	69.52%	65.23%	68%	85%

tion is effective when the code is used for drawing a large set of curves. Note that the parallel implementation of the rendering algorithm is not useful when we have a small number of curves (in the case of one single curve, the running

time increases with the processor number). Plotting the results from Table 2.(a) (see Figure 7), we can draw the conclusion that, if the curves set is large (i.e. of at least hundreds order), the efficiency of the parallel implementation decrease near linearly with the number of processors. The final conclusion is that the efficiency of the proposed algorithm increases with the number of curves and decreases with the number of processors.

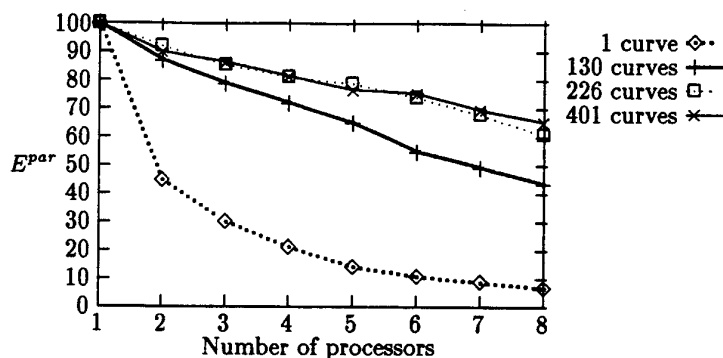


Fig. 7. Algorithm implementation efficiency relative to the number of processors and the number of curves

Experiment 3: Communication overhead in parallel and distributed implementations. The time results and the efficiency results of the code implemented on the transputer machine with the ones of the code implemented on the distributed system were compared. The results, for the case of 3 processors, are presented in Table 2.(b) (we have obtain also efficiency values, for the distributed system with $p = 2$ processors, between 69% for Figure 6 to 88% for Figure 5 and, for $p = 3$, between 53% to 82%). The difference are mainly due to the different rates of communication procedures. The efficiency results on the distributed system are encouraging the utilization of such system in rendering problems.

Experiment 4: Quality of the image produced by the proposed rendering algorithm. Different sets of quadrics have been considered and the resulting images and running times were compared with the ones produced using other rendering techniques: ray-tracing, depth-sort and z-buffer algorithms for polyhedral approximations. Due to some shading techniques, the images produced with the ray-tracing algorithm (with the POV-Ray's implementation, for example) are more realistic than those produced with the proposed rendering algorithm, but the running times are in favor of the second algorithm. Depth-sort technique is impracticable in the case of a large number of polygons of the approximate model (as the model from Figure 6 requests, for example). In order to create an image similar to that of the sphere from Figure 4, we have use the (sequential) z-buffer algorithm applied to a polyhedral model with 14400 vertex and 141161 polygons; although the discretization was very fine, one can distinguish the contours of each polygon (the same remark can be made for the model from Figure 8). The running times are also advantaging the new rendering algorithm.

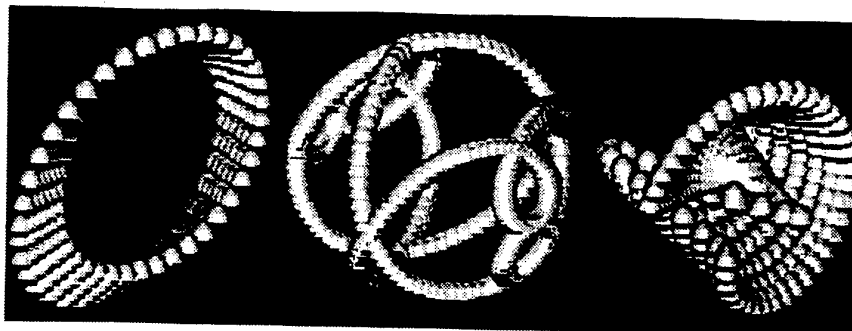


Fig. 8. Partial image produced by one processor in an implementation of the classic parallel z-buffer algorithm [5], applied to a polyhedral model associated to the set of quadrics from Figure 6.(a), with a data nonadaptive strategy, and with a static assignment of the processors (the final image, which has 206145 vertex, with 81 vertex for a sphere, and 196097 polygons, was created in four times more time than for the parallel rendering algorithm which treats 233993 section curves; the curve coefficients are more convenient to store than the polygonal informations). This half-image is produced much faster than the other half due to an load imbalance of the two processors.

References

1. Bresenham, J.: A Linear algorithm for incremental digital display of circular arcs. *Comm. ACM* **20** (1977), No. 2, 100-108
2. Elber, G., Cohen, E.: Adaptive isocurve-based rendering for freeform surfaces. *ACM Transactions on Graphics* **15** (1996), No. 3, 249-263
3. Fellner, D., Helmberg, C.: Robust rendering of general ellipses and elliptical arcs. *ACM Transactions on Graphics* **22** (1993), No. 3, 251-276
4. Foley, J.D., van Dam, A., Feiner S.K., Hughes J.F.: *Computer Graphics. Principles and Practice*, Second Edition. The Systems Programming Series, Addison Wesley Publishing Company, New York, 1992
5. Green, S.: *Parallel processing for computer graphics*. The MIT Press, Cambridge, 1991
6. Huang, J., Banissi, E.: An improved parallel circle-drawing algorithm, *IEEE Computer Graphics and Applications* **1** (1997) 40-41
7. Laporte H., Nyiri E., Froumentin M., Chaillou C.: A graphics system based on quadrics. *Computers and Graphics* **19** (1995) No. 2, 251-260
8. Petcu, D., Cucu, L.: Plotting conics and quadrics on a distributed computational system. *Annals of Timisoara's University, Mathematics-Computer Science Series* **33** (1995), No. 2, 221-231
9. Whitman, S.L.: Dynamic load balancing for parallel polygon rendering. *IEEE Computer Graphics and Applications*, **7** (1994), 41-48.
10. Wright, W.E.: Parallelization of Bresenham's line and circle algorithms. *IEEE Computer Graphics and Applications* **10** (1990), No.5, 60-67.
11. Wright, W.E.: Parallel algorithms for generating the raster representation of straight lines and circles. *Journal of Parallel and Distributed Computing* **11** (1991), 170-173
12. Yagel, R., Machiraju, R.: Data-parallel, volume-rendering algorithms. *The Visual Computer* **11** (1995), No. 6, 319-338.

Efficient Parallelization Approaches for the SAI Representation

A. Sánchez¹, S. Campos¹ and A. Rodríguez²

¹ Dpto. de Lenguajes y Sistemas Informáticos e Ingeniería de Software

² Dpto. de Tecnología Fotónica

Universidad Politécnica de Madrid

Campus de Montegancedo, s/n

28660 Boadilla del Monte (Madrid), Spain

Abstract. Discrete spherical representations have shown to be appropriate for modeling and recognizing 3D objects. The SAI (Simplex Angle Image) is an evolved spherical representation with several invariance properties, which enables the possibility to work with non-convex objects. This paper describes and compares the efficiency of parallelization strategies for the SAI using both a distributed (message-passing) and a shared memory approach. Several experiments have been carried out on workstations and on a multiprocessor. For each experiment, we have considered different sizes of files corresponding to 3D scattered points representing objects, a variable degree of tessellation in the unit Gaussian sphere, and a different number of working processes.

1 Introduction

One of the main objectives of a vision system is to achieve a good way to represent and model the acquired data. Nowadays, it is assumed that a good modeling system should have at least the following properties ([1],[2],[3]):

- Expressive richness.
- Stability (against errors or presence of noise in the input data).
- Treatment of occlusions.
- Physical attributes of the object's shape.
- Efficiency.

Among the different approaches of modeling 3D objects from real data (for example see [4],[5],[6],[7],[8]), a promising one is the model based on the Gaussian sphere proposed by Ikeuchi *et al.* ([9]-[16]). This model involves the definition of an homeomorphism from the input 3D cloud of points to a regular triangular mesh obtained by the subdivision of an icosahedron of radius unity with a previously fixed degree of resolution. The subdivision procedure constructs a surface similar to the approximation of the sphere surface with radius one by triangular facets. Each one of the nodes of the surface of the sphere stores the details of the local geometry that has the input data on the corresponding node

of the deformed mesh. Ikeuchi's model stores the Simplex Angle Image (SAI). The representation obtained by this procedure can be used on systems like object prototyping in CAD/CAM environments, sensorial perception equipments or object recognition systems.

This paper proposes several parallelization algorithms for the computation of the SAI using a shared memory and a distributed memory (MPI based) approaches. This work focuses on the experimental results and on its interpretation in relation with the factors involved in the parallel algorithms. The rest of the paper is organized as follows. Section 2 introduces the notation and outlines the steps describing the sequential algorithm. Descriptions and pseudocode corresponding to considered parallel strategies are given in Section 3. Experimental results and a global comparison of studied SAI algorithms are shown in Section 4. Finally, Section 5 presents the conclusions and gives some remarks on our future work.

2 Description of the SAI Algorithm.

This section summarizes the notation for the SAI and describes the main steps in the sequential algorithm.

2.1 Notation

The following terms will be introduced to be used in the rest of the paper:

SAI: Simplex Angle Image. The SAI angle represents the value of the curvature between the edge formed by one node of the regular mesh and one of his neighbors, and the plane formed by the node and the other two neighbors, once the regular mesh is adjusted to the 3D input data. The corresponding spherical representation, storing the SAI angle for each one of its nodes is called SAI. This representation is invariant by translation and scaling of the original object.

Acronyms for the SAI phases are:

CRE: Creation of the regular mesh at level of resolution specified.

CLP: Computation of the closest input point for each node of the regular mesh.

DEF: Deformation of the regular mesh in order to adjust the shape to the input 3D points.

SAIA: Computation stage of the SAI angle for each node of the Gaussian sphere.

n: Number of 3D points of the input data. In this work, we consider objects with a maximum of 83.171 points.

RD: Resolution degree of the regular mesh. In our experiments, we consider a maximum level of resolution equal to 5.

m: Number of nodes of the regular mesh. This number depends on the *RD* according to the following expression: $m = 20 \cdot 4^{RD}$, where $RD = 0, 1, 2, \dots$

NP: Number of processes considered in the parallel implementations. In order to have an uniform workload among processes, this number must divide 20 (number of facets of the initial approximation corresponding to the spherical mesh).

2.2 Sketch of the Algorithm

A sketch of the sequential algorithm for the definition of the SAI corresponding to an object, given by a file with 3D points obtained from range data, follows.

1. Construct the initial regular tessellated sphere which wraps around the set of 3D points defining the object. This involves three substeps: approximate the sphere by a 20-face icosahedron; tessellate recursively each one of its faces into n small triangular faces, where n ($n = 4^k, k = 1, 2, \dots$) and define the final tessellation by taking the dual of the previous triangulation, yielding a geodesic dome with the same number of nodes.
2. Determine for each node (face) of the geodesic dome, the 3D closest point in the data file corresponding to the object being modeled. This is the most time-consuming stage of the SAI algorithm, and its time complexity is $O(mn)$.
3. Perform an iterative deformation of the geodesic dome while the average sum of local errors exceeds a fixed threshold. Every local error is defined by the distance between a node of the sphere and its corresponding closest point, as determined in previous step. Each node is deformed to match the object, according to an approximation force F_o and a curvature force F_g , both defined for the actual position P_t of each node at time t . The new position of the node at time P_{t+1} is given by: $P_{t+1} = P_t + F_o + F_g + d(P_t - P_{t-1})$, where d represents the damping coefficient affecting the rate of convergence.
4. Compute the discrete curvature (simplex angle) for each node of the deformed mesh. Finally, map each simplex angle onto the corresponding node of the regular mesh, obtained at the end of the first step. The resulting structure is called Simplex Angel Image (SAI).

By analyzing the stages in the SAI representation, we found that it can be efficiently parallelized. Some remarks have to be taken into account: local nature of the deformation, completion condition for the deformation depends on a global error, and absence of a node topology which could define independent deformation regions on the mesh. These facts determine two implicit synchronization phases, which bound the improvements in the parallelization.

3 Parallel Strategies

Several parallelization strategies have been considered for the SAI. Implementations have been carried out using a message-passing interface standard (MPI) and using shared memory primitives on several workstations and on a multiprocessor with ten processors.

MPI represents the first attempt to standardize the communication library for distributed- memory computing systems ([17],[18]). One of the major goals of MPI is to provide a widely portable and ease-of-use programming library without sacrificing the performance. Considered parallel implementations for the SAI using MPI follow. In all these strategies, an equivalent data workload for all working processes (on the same or different machines) is achieved.

Collective communication MPI: It involves simultaneously a group of processes at a time (point-to-point communication involves only two processes). The information corresponding to complete nodes in the sphere is communicated among processes which work with contiguous data-dependent portions or the sphere. The following MPI primitives have been used in our implementation: **BROADCAST**, **ALLGATHER** and **REDUCE**. A sketch of the deformation step corresponding to this strategy is:

```

FOREACH process DO
  WHILE global_average_error > threshold DO
    Compute in parallel the new position of each node
    Compute in parallel the positions corresponding to the
      nodes of the "portion" of the spherical mesh
      associated with each process.
    Compute in parallel the local error for each node
    COMMUNICATION and SYNCHRONIZATION (MPI_ALLGATHER):
      local spherical "portion" of sphere corresponding to
      each process
    GLOBAL REDUCTION (MPI_REDUCE): local errors of
      processes are transformed into a global average
      error
  END
END

```

Optimized collective communication MPI: It differs from previous strategy in the fact that only the centers of nodes in the sphere are communicated among dependent processes. These values are the only necessary ones to execute a new iteration in the deformation step.

Shared-memory: Another different parallel approach is based on the use of shared memory primitives of a multiprocessor architecture. This strategy supports the creation of shared data accessible from every process. Thus, implicit communication is performed by global memory accesses. Semaphores are used to restrict the access to shared data and as a synchronization tool. In this implementation two shared data are defined: the spherical mesh (represented as an array of nodes, where each node stores the actual position of its center, the actual position of its three neighbor nodes and some other local information), and the global average error at each iteration in the deformation. A sketch of the deformation loop corresponding to this strategy is:


```

FOREACH process DO
  WHILE global average error > threshold DO
    compute in parallel the new position of each node
    SYNCHRONIZATION: actualize the global spherical mesh
    evaluate in parallel the local error for each node
    SYNCHRONIZATION and MUTUAL EXCLUSION: compute global
      average error
  END
END

```

4 Experimental Results

All the SAI strategies (sequential, shared memory, collective MPI and optimized collective MPI) considered in this paper have been implemented on several workstations (SGI5 175 MHz IP21 processor) and on a multiprocessor with ten processors (SGI5 200 MHz IP 19 processor). Other parallel MPI-based algorithms have also been implemented (i.e. point-to-point blocking and point-to-point non-blocking communication schemes) but the obtained execution times were considerably worse and have been discarded. The following subsections show the visual results corresponding to surface reconstruction of two objects used in our experiments, several figures corresponding to each one of studied SAI algorithms, and two tables which globally compare the performance of all strategies.

4.1 Input data and surface approximation

Our surface representation is a discrete connected mesh that is homeomorphic to a sphere. The connectivity of the mesh is such that each node has exactly three neighbors, and the total number of mesh nodes depends on its resolution degree. Given a set of 3D data points from range images, captured by a range laser and a 3D manual digitizer, the mesh representation is constructed as explained in Subsection 2.2. The mesh is always a closed surface. Figs. 1 and 2 show two examples of object surface reconstruction from 3D data. In our experiments, several objects with different shape and resolution have been considered. For the sake of simplicity, figures and tables are only referred to three objects: object obj1(315 points), object obj2(2928 points) and object obj3 (28656 points). The figures which refer to experiments that use only one object are referred to obj3, which is the most dense one. We should also point out that the deformable mesh is more suitable for representing rather smooth and convex objects.

4.2 Sequential algorithm

Figs. 3 and 4 present experimental results corresponding to the sequential implementation for the SAI model. Fig. 3 shows the percentage of execution time spent on each phase of the algorithm as a function of the mesh resolution degree. Fig. 4 presents the percentage of execution time for different object resolutions as a function of each algorithm stage.

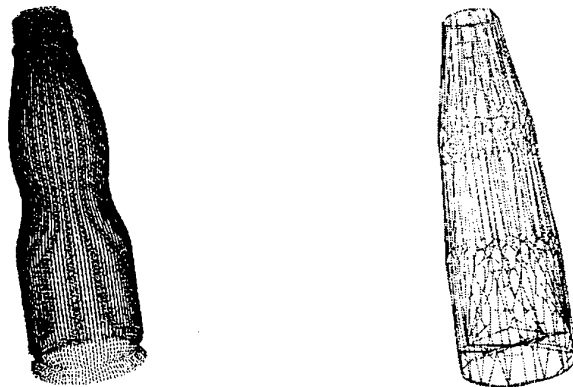


Fig. 1. (a) Original scattered point set of a bottle; (b) Deformed mesh adjusted to the original point set of the bottle.

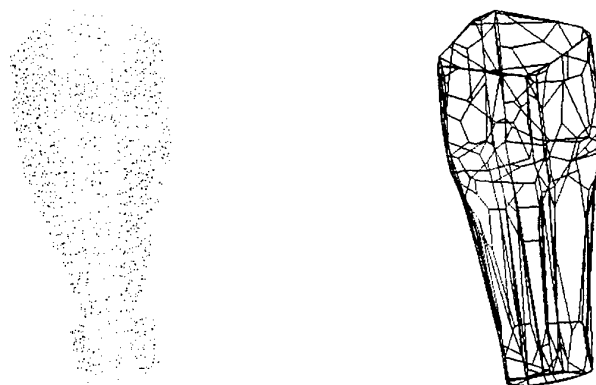


Fig. 2. (a) Original scattered point set of a synthetic object; (b) Deformed mesh adjusted to the original point set of this geometrical object.

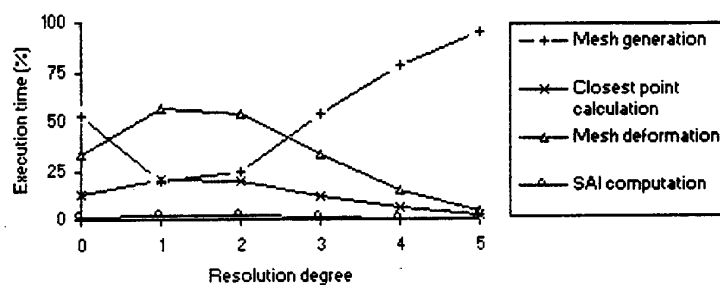


Fig. 3. Execution time of the sequential algorithm for obj3 by varying the resolution degree.

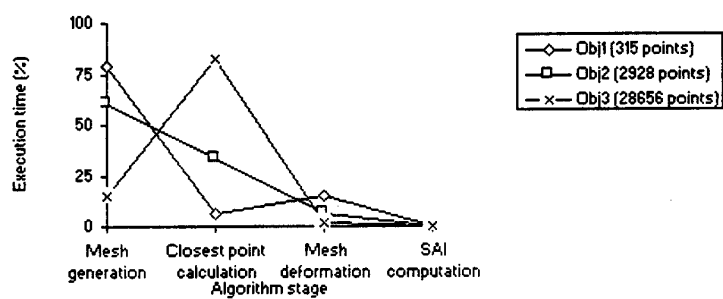


Fig. 4. Percentage of execution time corresponding to each algorithm stage for different objects in the sequential version.

4.3 Shared Memory

Figs. 5 and 6 illustrate the behavior of the shared memory version of SAI. The parameter NP must divide the size of initial sphere approximation (20 triangular facets), and we have plotted the representations for the values $NP=1,2,5,10$ and 20. In Fig. 5, note that the total execution time (expressed in logarithm scale) corresponding to $NP=5$ is worse with a low resolution degree but it improves as RD increases. Best results for high resolution are obtained using more processes. Fig. 6 shows, for a resolution $RD=3$, how the percentage of execution time decreases as the number of processes increases. It is interesting to remark that CLP stage consumes more than 85% of execution time when $NP=1$ or $NP=2$.

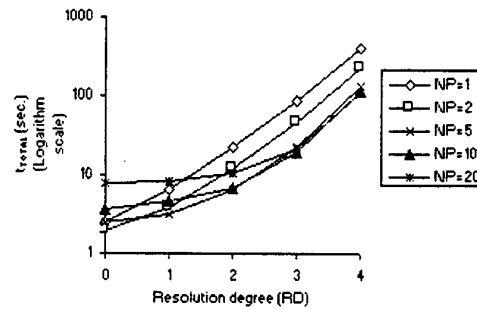


Fig. 5. Execution time of obj3 by varying the resolution degree for different number of working processes using the shared memory algorithm.

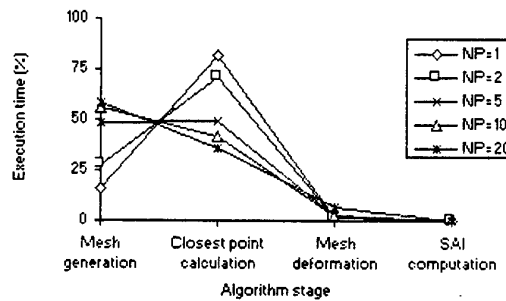


Fig. 6. Distribution of execution time among the algorithm stages for different number of processes using the shared memory version ($RD=3$).

4.4 Collective Communication MPI Algorithm

Figs. 7 and 8 are respectively equivalent to Figs. 5 and 6, but using message passing (MPI) concurrency. We can see in Fig. 7 that the total execution times for different number of processes are worse than the correspondent values of the shared memory algorithm (compare with values of Fig. 5). Similar behavior is observed in Fig. 8 (now, we have considered $RD=4$) with relation to Fig. 6. A remarkable fact in Fig. 8 is the high percentage of mesh deformation phase (next to 50 %) for $NP=10$, which is mainly due to the increase of communication among processes.

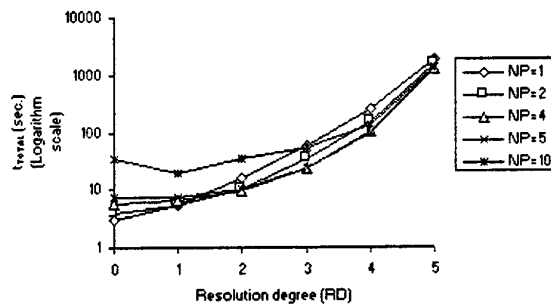


Fig. 7. Execution time of obj3 by varying the resolution degree for different number of working processes using the collective MPI version.

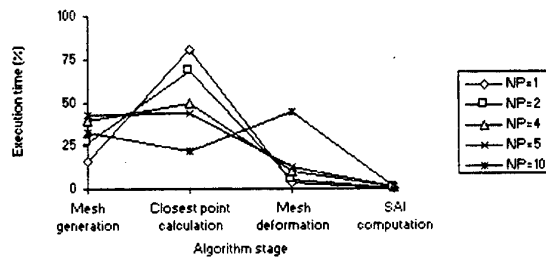


Fig. 8. Distribution of execution time among the algorithm stages for different number of processes using the collective MPI version (obj3, with $RD=4$).

4.5 Optimized Collective Communication MPI Algorithm

Figs. 9, 10 and 11 show the effect of the optimization in the previous collective MPI strategy. As pointed out in Section 3, this strategy significantly reduces the amount of data which are communicated among processes (only the information of a point is passed to neighbor nodes, instead of the whole information corresponding to a node structure). With this approach, as shown in Fig. 11, the execution time of each algorithm phase is reduced approximately by a factor of three with respect to the other considered MPI strategy (except for the mesh generation stage).

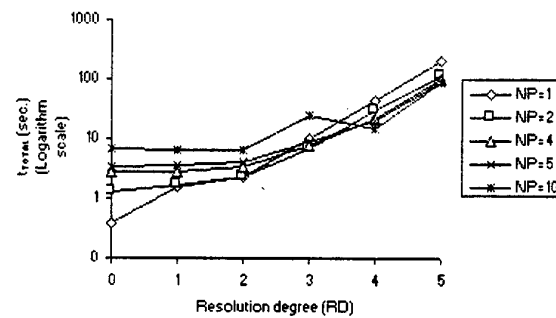


Fig.9. Execution time of obj3 by varying the resolution degree for different number of working processes using the optimized MPI version.

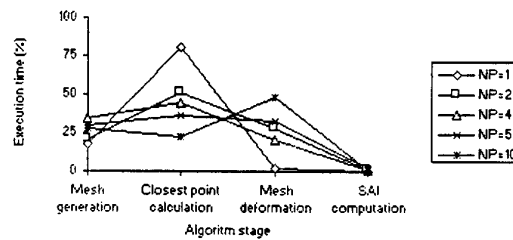


Fig.10. Distribution of execution time among the algorithm stages for different number of processes using the optimized MPI version (obj3, with $RD=4$).

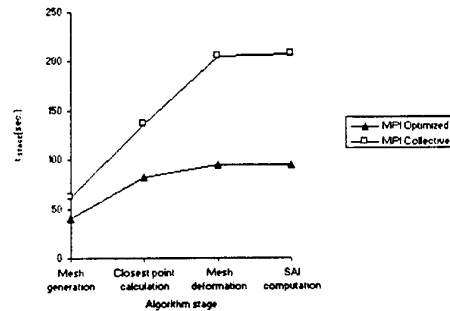


Fig.11. Comparison between collective and optimized MPI strategies using obj3 (28656 points), $NP=5$ and $RD=5$.

4.6 Global comparison

The graph in Fig. 12 compares the total execution time (t_{TOTAL}) as a function of the resolution degree for the different implemented SAI algorithms. The use of a logarithm scale to represent t_{TOTAL} , allows to better display the performance trends. A consequence of this figure is that for the considered object (obj3, 28656 points) when the resolution degree increases significantly, a better performance is achieved for the optimized collective communication MPI strategy (see this tendency for $RD=4$).

Table 1 expresses a relation of the execution time with respect to n (considered objects sizes represent different orders of magnitude, and can be approximately determined by multiplying the size of previous object by 10), the four main SAI stages and the four considered algorithms. From the overall time results, we note that, with independence of the object size, for the stages of mesh generation and closest point calculation, the optimized MPI algorithm gives the best results. By the other side, the shared memory strategy behaves better for the stages of mesh deformation and the calculation of the SAI angle. It is interesting to remark that the best execution times for the closest point calculation stage using optimized MPI algorithm are due to the lack of interaction among processes when calculating 3D distances (each process has its own copy of the 3D data), while in the shared memory algorithm there is an internal synchronization of processes when accessing to the unique copy of 3D data. The best execution times for the mesh deformation stage using the shared memory algorithm are due to the lack of exchange of data (except the local error for each node), while in the optimized MPI algorithm a global exchange of their own portion of the spherical mesh among working processes is required (MPI_ALLGATHER primitive, see Section 3).

To study in depth the behavior in the two most time-efficient algorithms (shared memory and optimized collective communication MPI, respectively), they have been compared with respect to an object with 83,171 points. Table 2 shows the best execution results as a function of the resolution degree for both

algorithms. Each execution time is accompanied with the number of software processes involved in it. Experimental results show a similarity in resolution degree of 0, 1 and 2; a better performance of the optimized communication MPI strategy for resolutions of 3 and 4; and a slight advantage of shared memory for resolution of 5. These global execution times are mainly explained by the influence of CLP calculation phase (the most time-consuming one in the algorithm) which has a better behavior in the shared memory algorithm as the resolution degree (RD) grows.

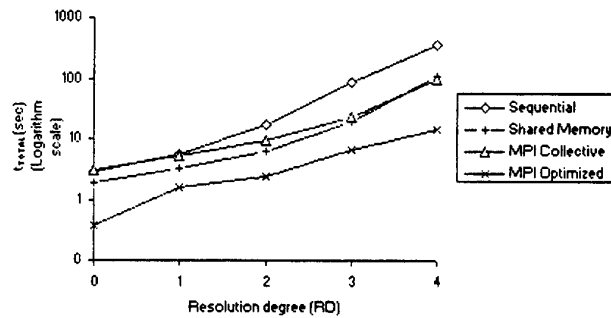


Fig. 12. Execution time versus resolution degree for the different SAI algorithms applied to the object obj3.

Table 1. Global comparison of execution times (in seconds) for the SAI algorithm stages with considered objects (obj1=315 points, obj2=2928 points, and obj3=28656 points), $RD=3$ and $NP=5$.

		Sequential	Shared Memory	MPI Collective	MPI Optimized
obj1	CRE	4.038	3.985	3.380	0.746
	CLP	0.944	0.189	0.060	0.001
	DEF	5.569	0.978	46.582	24.031
	SAIA	0.073	0.016	0.653	0.174
obj2	CRE	4.049	4.046	2.919	0.297
	CLP	8.406	1.704	1.360	0.052
	DEF	1.680	0.971	5.574	2.486
	SAIA	0.073	0.016	0.120	0.075
obj3	CRE	5.068	5.012	4.669	0.908
	CLP	81.825	16.307	12.943	5.119
	DEF	1.806	0.983	5.791	0.514
	SAIA	0.074	0.017	0.111	0.140

Table 2. Comparison of shared memory and optimized collective MPI strategies for an object with 83.171 points.

Resolution degree	Shared Memory		MPI Optimized	
	t_{TOTAL} (sec.)	NP	t_{TOTAL} (sec.)	NP
0	0.412	4	0.527	2
1	0.556	5	0.865	4
2	1.033	16	1.493	5
3	7.113	10	4.239	5
4	20.074	16	16.143	8
5	99.027	16	102.328	8

5 Conclusions and Future Work

Results of several parallel implementations of the SAI model computation have been presented in this paper. The parallel strategies, one shared memory and two MPI-based algorithms, have been compared with the sequential version. Due to the different nature of each stage of the SAI algorithm, shared memory implementation gives better execution results than MPI implementations when less communication is involved on the computation, i.e. on stages DEF and SAIA. On the other hand, when the locality of the computations allows the distribution of the workload between all the available processes, results are favourable to the MPI implementations, i.e. on stages CRE and CLP.

It should be mentioned that the development of the MPI implementations has been easier than using the shared memory primitives of a high level language, due to the fact that the MPI distribution (we have employed CHIMP vs. 2.0) allows the use of powerful collective parallel programming primitives than the standard libraries available for shared memory concurrency.

A future improvement is focused on stage CLP, whose execution time could perhaps be reduced using 3D computational geometry techniques. Another development will be the parallelization of the sequential version using the SGI parallel compiler in order to compare the results with that provided by a commercial compiler.

Actually, the surface reconstruction algorithm reduces the approximation error when convex objects are considered. We think this error can be reduced for non-convex objects when the points of the considered object, closest to each node of the spherical mesh, are recomputed in each iteration during the deformation.

6 Acknowledgments

This research work has been supported in part by the Spanish CICYT under the grant TAP94-0305-C03-02.

The authors gratefully acknowledge the access to the T3D at EPCC in Edinburgh (UK). Margarita Bachiller from the U.N.E.D. has helped in the initial sequential version of the SAI algorithm. Juan López Coronado and Jaime Gómez

from the University of Valladolid and Javier Durán have provided some of the data used to test the system. Domingo López and Pablo Aguiló also helped in the visualization of surface approximations of point clouds. The help of all of them is gratefully acknowledged.

References

1. T. J. Fan: Describing and Recognizing 3-D objects Using Surface Properties. Springer Verlag (1990).
2. E. L. Grimson: Introduction of the Special Issue on Interpretation of 3D Scenes. IEEE Trans. on PAMI. **14**, 2, (1992), 97-98.
3. S. J. Dickinson, R. Bergevin, I. Biederman, J. Eklundh, R. Munck-Fairwood and A. Pentland: The Use of Geons for Generic 3D Object Recognition. Proc. 13th Int. Conf. on Artificial Intelligence, Chambéry, Francia, (1993) pp. 1693-1699.
4. D. Terzopoulos y D. Metaxas: Dynamic 3D Models with Local and Global Deformations: Deformable Superquadrics. IEEE Trans. on PAMI. **13**, 7, (1991), 703-714.
5. S. Kumar, S. Han, D. Goldgof and K. Bowyer: On Recovering Hyperquadrics from Range Data. IEEE Trans. on PAMI. **17**, 11, (1995), 1079-1083.
6. C. Dorai, G. Wang, A. K. Jain and C. Mercer: Registration and Integration of Multiple Object Views for 3D Model Construction. IEEE Trans. on PAMI. **20**, 1, (1998), 83-89.
7. S. J. Dickinson, D. Metaxas and A. Pentland: The Role of Model-Based Segmentation in the Recovery of Volumetric Parts From Range Data. IEEE Trans. on PAMI. **19**, 3, (1997), 259-267.
8. C. Oblonsek and N. Guid: A Fast Surface-Based Procedure for Object Reconstruction from 3D Scattered Points. Computer Vision and Image Understanding, **69**, 2, (1998), 185-195.
9. Ikeuchi K.: Recognition of 3-D objects using the Extended Gaussian Image. Proc. IJCAI81, (1981), 595-600.
10. Kang S. B. and Ikeuchi K.: The Complex EGI: A New Representation for 3-D Pose Determination. IEEE Trans. on PAMI. **15**, 7, (1993), 707-721.
11. Delingette H., Hebert H. and Ikeuchi K.: A Spherical Representation for the Recognition of Curved Objects. Proc of the ICCV, Berlin, (1993), 103-112
12. Higuchi K., Delingette H., Hebert M. and Ikeuchi K.: Merging Multiple Views Using a Spherical Representation. Proc. of Second CAD-Based Vision Workshop. IEEE, (1994), 124-131
13. Ikeuchi K. and Hebert M.: Spherical Representations: from EGI to SAI Proc. Intl. NSF-ARPA Workshop "Object Representation in Computer Vision", (1995), 327-345.
14. Ikeuchi K. and Hebert M.: Spherical Representations: From EGI to SAI. Proc. Intl. NSI-ARPA Workshop. (1995), 327-345
15. K. Ohba and K. Ikeuchi: Detectability, Uniqueness, and Reliability of Eigen Windows for Stable Verification of Partially Occluded Objects. IEEE Trans. on PAMI. **19**, 9, (1997), 1043-1048.
16. H. Shum, M. Hebert, K. Ikeuchi and R. Reddy: An Integral Approach to Free-Form Object Modeling. IEEE Trans. on PAMI. **19**, 12, (1997), 1366-1370.
17. MPI Forum: MPI: A Message-Passing Interface Standard. (1995)
18. Natawut Nupairoj and Lionel M. Ni: Performance Evaluation of Some MPI Implementations on Workstations Clusters. Proc. of the 1994 Scalable Parallel Libraries Conference (SPLC'94), (1994), 98-105.

Parallel Implementations of Morphological Connected Operators Based on Irregular Data Structures

Christophe Laurent^{1 2} and Jean Roman²

¹ France Telecom - CNET/DIH/HDM, 4, Rue du Clos Courtel
35 512 Cesson Sévigné Cedex, France

² LaBRI UMR CNRS 5800, 351, Cours de la Libération
33 405 Talence Cedex, France

Abstract. In this paper, we present parallel implementations of connected operators. These kind of operators have been recently defined in mathematical morphology and have attracted a large amount of research due to their efficient use in image processing applications where contour information is essential (image segmentation, pattern recognition ...). In this work, we focus on connected transformations based on geodesic reconstruction process and we present parallel algorithms based on irregular data structures. We show that the parallelization poses several problems which are solved by using appropriate communication schemes as well as advanced data structures.

1 Introduction

In the area of image processing, mathematical morphology [1] has always proved its efficiency by providing a geometrical approach to image interpretation. Thus, contrary to usual approaches, image objects are not described by their frequential spectrum but by more visual attributes such as size, shape, contrast ...

Recently, complex morphological operators, known as *connected operators* [2], have been defined and become today increasingly popular in image processing because they have the fundamental property of simplifying an image without corrupting contour information. Through this property, this class of operators can be used for all applications where contour information is essential (image segmentation, pattern recognition ...).

In this paper, we focus on the parallelization of connected operators based on a geodesic reconstruction process [3] aimed at reconstructing the contours of a reference image from a simplified one. In spite of their efficiency in the sequential case, these transformations are difficult to parallelize efficiently, due to complex propagation and re-computation phenomena, and thus, advanced communication schemes and irregular data structures have to be used in order to solve these problems.

The proposed parallel algorithms are designed for MIMD (*Multiple Instruction Multiple Data*) architectures with distributed memory and use a message passing programming model.

This paper is organized as follows : section 2 introduces the theoretical foundations of morphological connected operators and presents the connected transformations based on geodesic reconstruction process. Section 3 makes a survey of the most efficient sequential implementations which are used as starting point of the parallelization. In section 4, we propose to detail all parallel algorithms for morphological reconstruction and the experimental results obtained on a IBM SP2 machine are presented in section 5. Finally, we conclude in section 6.

2 Overview of Morphological Connected Operators

This section presents the concept of morphological connected operators and we invite the reader to refer to [2,4,5] for a more theoretical study.

In the framework of mathematical morphology [1], the basic working structure is the *complete lattice*. Let us recall that a complete lattice is composed of a set equipped with a total or partial order such that each family of elements $\{x_i\}$ possesses a supremum $\vee\{x_i\}$ and an infimum $\wedge\{x_i\}$. In the area of grey-level image processing, the lattice of functions (where the order, \vee and \wedge are respectively defined by \leq , *Max* and *Min*) is used.

Following this structure, the definition of grey-level connected operators has been given in [2,4] by using the notion of *partition of flat zones*. Let us recall that a partition of a space E is a set of connected component $\{A_i\}$ which are disjoint ($A_i \cap A_j = \emptyset \forall i \neq j$) and the union of which is the entire space ($\cup A_i = E$). Each connected component A_i is then called a *partition class*. Moreover, a partition $\{A_i\}$ is said to be *finer* than a partition $\{B_j\}$ if any pair of points belonging to the same class A_i also belongs to a unique partition class B_j . Finally, the flat zones of a grey-level function f are defined by the set of connected components of the space where f is constant. In [2], the authors have shown that the set of flat zones of a function defines a partition of the space, called the *partition of flat zones of the function*.

From these notions, a grey-level connected operator can be formally defined as follows :

Definition 1. An operator Ψ acting on grey-level functions is said to be connected if, for any function f , the partition of flat zones of $\Psi(f)$ is less fine than the partition of flat zones of f .

This last definition shows that connected operators have the fundamental property of simplifying an image while preserving contour information. Indeed, since the associated partition of $\Psi(f)$ is less fine than the associated partition of f , each flat zone of f is either preserved or merged in a neighbor flat zone. Thus, no new contour is created.

Following this principle, we can decompose a grey-level connected operator in two steps : the first one, called *selection step*, assesses a given criterion (size, contrast, complexity ...) for each flat zone and from this criterion, the second step called *decision step* decides to preserve or merge the flat zone. A large number of grey-level connected operators can thus be defined, only differencing in

the criterion used by the selection step. For a presentation of different connected operators, refer to [6].

In this paper, we focus on connected operators based on geodesic reconstruction process [3]. This kind of transformation is actively used in applications such as grey-level and color image and video segmentation [7-9], defect detection [10], texture classification [11] and so forth. The reconstruction process is based on the definition of the *elementary geodesic dilation* $\delta^{(1)}(f, g)$ of the grey-level image $g \leq f$ "under" f defined by $\delta^{(1)}(f, g) = \text{Min}\{\delta_1(g), f\}$ where $\delta_1(g)$ defines the elementary numerical dilation of g given by $\delta_1(g)(p) = \text{Max}\{g(q), q \in N(p) \cup \{p\}\}$ with $N(p)$ denoting the neighborhood of the pixel p in the image g .

The geodesic reconstruction of a *marker image* g with reference to a *mask image* f with $g \leq f$ is obtained by iterating elementary gray-level geodesic dilation of g "under" f until stability :

$$\rho(g | f) = \delta^{(\infty)}(f, g) = \delta^{(1)}(\delta^{(1)}(\dots(\delta^{(1)}(f, g))\dots)) \quad (1)$$

Figure 1 shows a 1D example of the reconstruction process $\rho(g | f)$ for which we can note that all contours of the function f are perfectly preserved after the reconstruction.

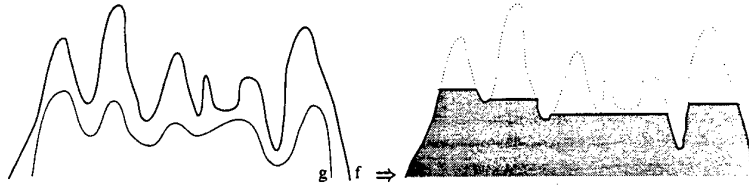


Fig. 1. 1D example of grayscale reconstruction of mask f from marker g

Following the technique used to obtain the marker image g , it is straightforward that a large number of connected operators can be defined. Two connected operators, based on this reconstruction process, are intensively used in literature. The first one, known as *opening by reconstruction*, has a size oriented simplification effect since the marker image is obtained by a morphological opening removing all bright objects smaller than the size of the structuring element. The second one, known as λ - *Max* operator, has a contrast oriented simplification effect since the marker image is obtained by subtracting a constant λ to the mask image ($g = f - \lambda$). Figure 2 shows these two transformations. From the *Cameraman* test image (see Figure 2(a)), Figure 2(b) shows the result of an opening by reconstruction $\rho(\gamma_{15}(f) | f)$ where $\gamma_n(f)$ denotes the morphological opening of the function f using a structuring element of size n . Figure 2(c) shows a λ - *Max* operator with $\lambda = 40$ denoted by $\rho(f - 40 | f)$.

Note that for each defined connected operator, a dual transformation can be obtained by reversing the *Max* and *Min* operator. We thus obtain a *closing*

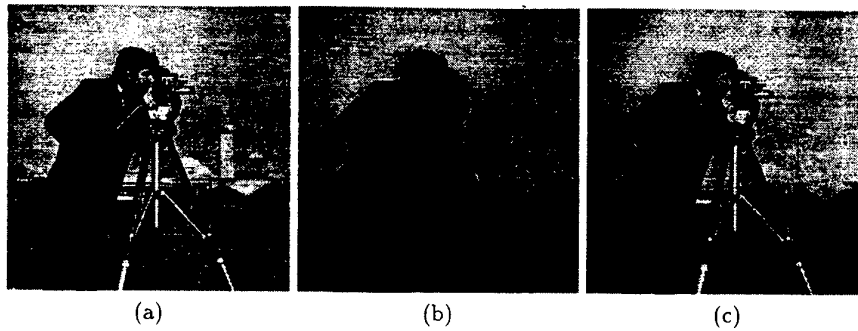


Fig. 2. Example of geodesic reconstruction on grey-level images

by *reconstruction* transformation, that removes all dark objects smaller than the structuring element, and a λ -*Min* transformation, that removes all objects which have a contrast higher than λ . Due to this duality relation, we only study in this paper the reconstruction process by geodesic dilation (see equation (1)).

3 Efficient Sequential Implementations of Geodesic Reconstruction

The most efficient algorithms for geodesic reconstruction have been proposed in [3]. In this paper, we only focus on implementations based on irregular data structures which consider the image under study as a graph and realize a breadth-first scanning of the graph from strategically located pixels [12]. These algorithms proceed in two main steps :

- detection of the pixels which can initiate the reconstruction process,
- propagation of the information only in the relevant image parts.

In the case of reconstruction by geodesic dilation (see equation (1)), we can easily show that a pixel p can initiate the reconstruction of a mask image f from a marker image g if it has in its neighborhood at least one pixel q such that $g(q) < g(p)$ and $g(q) < f(q)$.

Proposition 2. *In the reconstruction of a mask image f from a marker image g with $g \leq f$, the only pixels p which can propagate their grey-level value in their neighborhood verify : $\exists q \in N(p), g(q) < g(p)$ and $g(q) < f(q)$.*

Proof. Let p be a pixel such that $\exists q \in N(p), g(q) < g(p)$ and $g(q) < f(q)$.

We have $\delta^{(1)}(f, g)(q) = \text{Min}\{\delta_1(g)(q), f(q)\}$ with $\delta_1(g)(q) = \text{Max}\{g(t), t \in N(q) \cup \{q\}\}$.

Since $q \in N(p)$, we have $p \in N(q)$. Moreover, we know that $g(q) < g(p)$ and thus, q is not a fixed point of the transformation δ_1 eg $\delta_1(g)(q) > g(q)$.

On the other hand, since $g(q) < f(q)$, the transformation $\delta^{(1)}(f, g)$ has not reached stability at the location of q . As a result, the pixel q can receive a propagation from the pixel p .

To conclude the proof, it is straightforward that if $\forall q \in N(p), g(q) > g(p)$, the pixel p cannot propagate its grey-level value but receives a propagation from its neighborhood. \square

Note that a same pixel can propagate its grey-level value on a neighbor pixel and receive a propagation from another neighbor pixel.

Following this principle, two methods have been proposed in [3] to detect initiator pixels. The first one consists in computing the regional maxima of the marker image g which designate the set of connected components \mathcal{M} with a grey-level value h in g such that every pixel in the neighborhood of \mathcal{M} has a strictly lower grey-level value than h . In this case, initiator pixels are those located in the interior boundaries of regional maxima of g .

The second method is based on two scanning of the marker image g . The first scanning is done in a raster order (from top to bottom and left to right) and each grey-level value $g(p)$ becomes $g(p) = \min\{\max\{g(q), q \in N^+(p) \cup \{p\}\}, f(p)\}$ where $N^+(p)$ is the set of neighbors of p which are reached before p in a raster scanning (see Figure 3(a)). The second scanning is done in the anti-raster order and each grey-level value $g(p)$ becomes $g(p) = \min\{\max\{g(q), q \in N^-(p) \cup \{p\}\}, f(p)\}$ where $N^-(p)$ designates the neighbors of p reached after p in the raster scanning order (see Figure 3(b)). In this case, the initiator pixels are detected in the second scanning and are those which could propagate their grey-level value during the next raster scanning. These pixels p verify : $\exists q \in N^-(p), g(q) < g(p)$ and $g(q) < f(q)$.

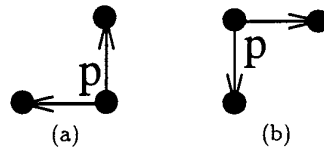


Fig. 3. Definition of $N^+(p)$ (a) and $N^-(p)$ (b) in the case of 4-connectivity

One can easily see that these two methods verify the general principle exposed in Proposition 2.

Once the initiator pixels are detected, the information has to be propagated by a breadth-first scanning. For this purpose, the breadth-first scanning is implemented in [3] by a queue of pixels represented by a FIFO (*First In First Out*) data structure. The initiator pixels are first inserted in the queue and the propagation consists in extracting the first pixel p from the queue and propagating its grey-level value $g(p)$ to all of its neighbors q such that $g(q) < g(p)$ and $g(q) < f(q)$. The grey-level of these pixels becomes then $g(q) = \min\{g(p), f(q)\}$

and the last operation consists in inserting these pixels q in the queue in order to continue the propagation. The reconstruction stops when the queue is empty.

4 Parallel Implementations of Geodesic Reconstruction

4.1 Preliminaries

In this paper, we are interested in the parallelization of morphological reconstruction based on geodesic dilation since by duality, reconstruction by geodesic erosion can be obtained by reversing the *Min* and *Max* operators in equation (1).

All proposed parallel algorithms are designed for MIMD (*Multiple Instruction Multiple Data*) architectures with distributed memory and use a message passing programming model. Based on this parallel context, we denote by p the number of processors and by P_i the processor indexed by i ($0 \leq i < p$).

The mask image f and the marker image g , for which the domain of definition is denoted by D , are splitted into p sub-images f_i and g_i defined on disjoint domains D_i ($0 \leq i < p$). For the sake of simplicity, we assume that the partitioning is made in a horizontal 1D-rectilinear fashion. Thus, if we suppose that f and g are of size $n \times n$, each processor P_i owns a mask sub-image f_i and a marker sub-image g_i , each of size $\frac{n}{p} \times n$. Note that all proposed algorithms can be extended to a 2D-rectilinear partitioning scheme without difficulty.

Moreover, in order to study the propagation property of its local pixels on non local pixels located in neighbor sub-images, each processor P_i owns two 1-pixel wide overlapping zones on its neighbor sub-images f_{i-1} and g_{i-1} (for $i > 0$), and f_{i+1} and g_{i+1} (for $i < p - 1$). From this extended sub-domain, denoted by ED_i , we denote by $LB(i, j)$ the local pixels located on the frontier between P_i and P_j that is $LB(i, j) = D_i \cap ED_j$, and by $NB(i, j)$ the non local pixels located on the frontier between P_i and P_j that is $NB(i, j) = ED_i \cap D_j$. Figure 4 shows all of these notations.

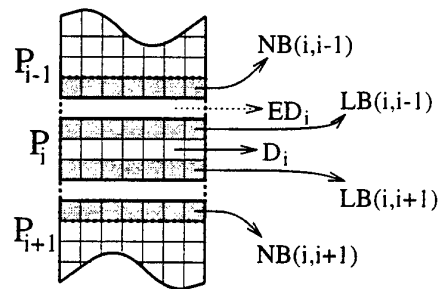


Fig. 4. 1D rectilinear data partitioning

On the other hand, we denote by $N_D(p)$ the neighborhood of the pixel p in the domain D and by $N_{D_i}(p)$ the neighborhood of the pixel p restricted to the sub-domain D_i .

Before presenting our parallel geodesic reconstruction algorithms, we would like to underline the difficulty of this parallelization and show the motivation of this work. As mentioned in section 2, geodesic reconstruction is not a locally defined transformation resulting in a complex propagation phenomenon, that makes it difficult to parallelize efficiently. If we consider the example of reconstruction by geodesic dilation (see equation (1)), we remark that a bright pixel can propagate its grey-level value to a large part of the image and no technique allows us to predict this phenomenon a priori. Now, in parallel implementations where initial images are splitted and distributed among processors, each processor proceeds reconstruction from its local data and thus, when a propagation phenomenon appears and crosses inter-processors frontier, it can generate the re-computation of a large number of pixels located in the sub-image that have received the propagation. Figure 5 illustrates this phenomenon with a 1D distribution in which the mask image f and the marker image g are distributed on two processors P_0 and P_1 . On this example, the processor P_1 owns a dark marker sub-image whereas the processor P_0 owns a bright marker sub-image (see Figure 5(a)). In a computational point of view, if P_1 entirely reconstructs its local image before taking into account the propagation messages sent by P_0 , it is straightforward that P_1 will have to re-compute a large number of pixels and the resulting total execution time will be slowed down (see Figures 5(b,c)). The motivation of this work is thus to adopt communication schemes adapted to this inter-processors propagation phenomenon and to propose efficient data structures limiting the re-computation phenomenon.

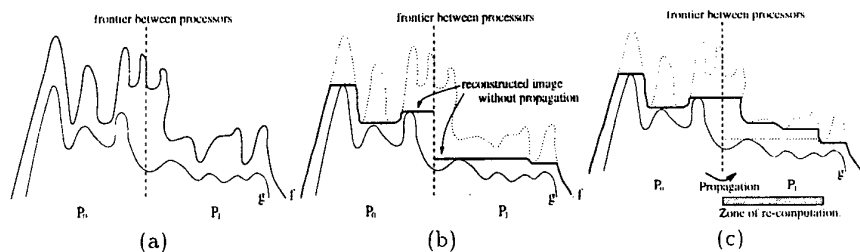


Fig. 5. Propagation and re-computation phenomena

In this paper, we only focus on parallel geodesic reconstruction algorithms based on irregular data structures represented by pixel queues because of their efficiency in the sequential case. As mentioned in section 3, sequential algorithms proceed in two steps : detection of initiator pixels and propagation of the information. One can easily remark that parallel algorithms which follow this principle are very irregular since on one hand, the number of detected initiator

pixels can differ on each processor and on the other hand, the propagation of information can differently evolve on each processor. Moreover, from this irregular nature, communication primitives, that have to take place in order to propagate information between processors, will be irregular since the number of sent and received propagations can differ on each processor.

4.2 Parallel Reconstruction Based on Regional Maxima

The parallel algorithm based on the use of marker image regional maxima proceeds in two steps. In the first one, all processors compute in parallel the regional maxima of the marker image g (for this step, refer to [13]) and produce a temporary image \mathcal{R} defined by :

$$\mathcal{R}(p) = \begin{cases} g(p) & \text{if } p \text{ belongs to a regional maxima} \\ 0 & \text{otherwise.} \end{cases}$$

From the image \mathcal{R} , each processor P_i initializes its local pixel queue F_i by inserting all pixels $p \in D_i$ located on internal boundaries of regional maxima :

$$F_i = \{p \in D_i, \mathcal{R}(p) \neq 0 \text{ and } \exists q \in N_D(p) \text{ such that } \mathcal{R}(q) = 0\}.$$

During the last step, the information is propagated through the image and each processor can then receive a propagation from its neighbor processors. These interactions are implemented by communication primitives and two methods can be proposed for this step : the *synchronous approach* in which communications are considered as synchronization points which regularly appear, and the *asynchronous approach* in which communications are exchanged immediately after the detection of propagation.

Synchronous Approach. In a first time, each processor P_i applies the reconstruction process starting from initiator pixels inserted in F_i after the computation of regional maxima. After the consumption of F_i , each processor P_i exchanges with its neighbors P_j ($j = i - 1, i + 1$) all pixels of g_i located in $LB(i, j)$. From the received data and the local ones, each processor P_i can then detect its local pixels that have received a propagation. These pixels p are those located in $LB(i, j)$ ($j = i - 1, i + 1$) and which have a neighbor q located in the neighbor sub-image g_j such that $g_j(q) > g_i(p)$ and $g_i(p) < f_i(p)$. These pixel values become then $g_i(p) = \min\{g_j(p), f_i(p)\}$ (see Figure 6). These pixels form a new set of initiator pixels and are thus inserted in turn in F_i . The reconstruction process can then be iterated with this new set.

This general scheme is iterated n_{it} times until a global stabilization, which is detected when no processor receives a propagation from its neighbors.

This approach follows a "divide and conquer" programming paradigm and can be qualified as semi-irregular since the local propagation step remains irregular because based on FIFO data structure but communications are performed in a regular and synchronized fashion.

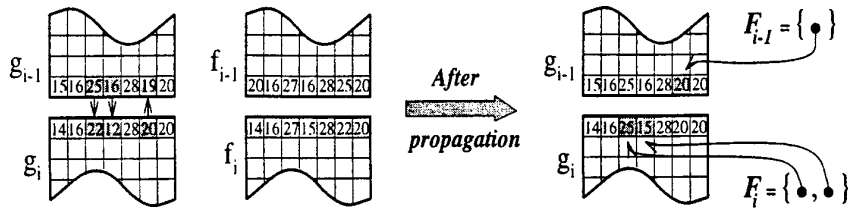


Fig. 6. Example of inter-processor propagation

The main disadvantage of this technique is to take into account the inter-processor propagations at latest, resulting in an important re-computation phenomenon. Moreover, a propagation starting on a processor P_i and going up to P_{i+k} ($k \in [-i, p-i-1]$) will be taken into account by P_{i+k} after k communication steps. Finally, due to the regular nature of the communications, no overlapping of computation and communications are proposed by this approach.

Asynchronous Approach. Contrary to the previous approach, inter-processor propagation messages are here exchanged as one goes along their detection in order to take them into account as soon as possible. This technique solves the problems posed by the synchronous approach since it reduces the amount of communications crossing the network and limits the re-computation phenomenon.

In this approach, the reconstruction stops when all processors have cleaned their local pixel queue and when all sent propagation messages have been taken into account by their recipients. In order to detect this stabilization time, a token is used reporting all sent propagation messages which have not yet been taken into account. This token is implemented by a vector T of size p in which $T[i]$ ($0 \leq i < p$) designates the number of messages sent to P_i and not yet received. Thus, the global stabilization is reached when all entries of T become nil.

In the same manner of the synchronous approach, when the grey-level value of a boundary pixel p is modified by a processor P_i , a message is sent to the corresponding neighbor processor P_j indicating the position x and the new grey-level value $g_i(p)$ of the modified pixel p . The processor P_j receiving a message $(x, g_i(p))$ of this kind takes into account the propagation effect of the pixel p on its own marker sub-image g_j . For this purpose, it scans all pixels q located in g_j and in the neighborhood of p ($q \in N_{D_j}(p)$) and it reports propagation on all of these pixels verifying $g_j(q) < g_i(p)$ and $g_j(q) < f_j(q)$. The grey-level value of these pixels becomes then $g_j(q) = \min\{g_i(p), f_j(q)\}$. The last step consists in inserting the pixel q in the queue F_j .

Finally, once a processor P_i has finished its local reconstruction ($F_i = \emptyset$), it enters in a blocking state until the global stabilization is reached. In this state, it can either receive a propagation message from a neighbor processor that have not yet terminated its reconstruction or a message for the token management.

In these synchronous and asynchronous irregular algorithms, the consumption order of the pixels during the propagation step directly depends on their insertion order in the FIFO, which depends in turn on the image scanning order in the initialization phase. Following this observation, we can remark that for a sequential as well as parallel geodesic reconstruction, some pixels of marker image can be modified more than once by receiving propagations from several pixels. In the case of reconstruction based on regional maxima, this problem appears when some pixels, which have received a propagation from a regional maxima \mathcal{M}_1 with a grey-level value h_1 , can also be reached by a regional maxima \mathcal{M}_2 with a grey-level value $h_2 > h_1$. Let us consider the example illustrated on Figure 7(a) where the marker image has two regional maxima \mathcal{M}_1 and \mathcal{M}_2 with a respective grey-level value of 10 and 20. For the sake of simplicity, we assume that the mask image has a constant grey-level value equal to 25. Suppose now that during the initiator pixels detection step, the pixels located in internal boundaries of \mathcal{M}_1 are inserted in the FIFO before the pixels located in internal boundaries of \mathcal{M}_2 . On this Figure, where arrows designate the propagations sense, each maximum is extended to its neighborhood. At the second step of propagation (see Figure 7(b)), the maximum \mathcal{M}_2 , that has a grey-level value higher than h_1 , begins to generate the re-computation of some pixels previously modified by the propagation phase initialized from \mathcal{M}_1 . As one goes along of iterations, each of maximum is spatially extended and all pixels located in the intersection of these extensions are re-computed (see Figure 7(c,d) where re-computation are designed by dark grey dashed square). At the end of the reconstruction, all pixels of marker image have a grey-value of 20 and thus, all pixels modified during the extension of \mathcal{M}_1 have been re-computed by receiving a propagation from \mathcal{M}_2 .

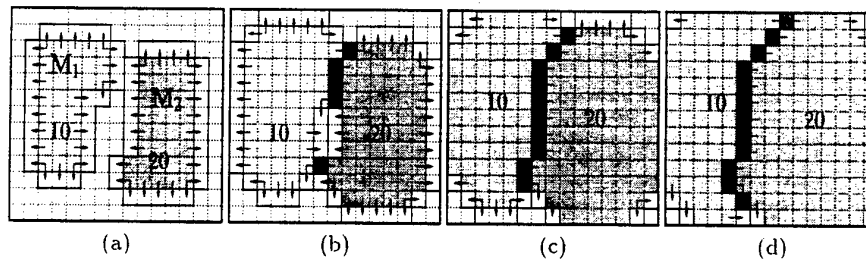


Fig. 7. Re-computation phenomenon for reconstruction based on regional maxima

It would be thus interesting to propose a data structure ensuring the modification unicity of each pixel. For this purpose, it is important to observe that in the case of reconstruction by geodesic dilation (see equation (1)), the propagation sense is always from bright pixels to dark ones, and thus we can affect a priority of reconstruction to each pixel, attached with its grey-level value. In the case of reconstruction by geodesic dilation of a mask image f from a marker image g , each pixel p will thus be inserted in the queue with a priority given

by $g(p)$. One can easily see that this technique cancels the re-computation phenomenon in the sequential case and limits it as far as possible in the parallel case. Moreover, the priority mechanism can be efficiently implemented by using hierarchical FIFO [14] with as much priority as grey-level values in the marker image.

From this new data structure, all proposed algorithms can be rewritten by only modifying the calls to FIFO management.

4.3 Hybrid Parallel Reconstruction

In the previous implementations, a large part of the reconstruction time is dedicated to the computation of regional maxima. To solve this problem, it has been proposed in [3] to detect initiator pixels from two scanings of marker image as explained in section 3. Following this principle, all techniques presented in the previous section and devoted to the propagation step can be applied here since the only modified step concerns with the detection of initiator pixels. Thus, the parallel algorithms based on this technique proceed in two steps :

- detection of initiator pixels from two scanings of marker sub-image,
- propagation of information in a synchronous or asynchronous way by using a classical or hierarchical FIFO.

5 Experimental Results

In this section, we present some experimentations of our parallel algorithms obtained on a IBM SP2 machine with 16 processing nodes by using the MPI (*Message Passing Interface*) [15] communication library. For these measures, four test images (*Cameraman*, *Lena*, *Landsat* and *Peppers*) have been used, each of size 256×256 . For *Cameraman* and *Lena* test images, a parallel opening by reconstruction has been measured for which the marker image has been obtained by a morphological opening of size 5 and 15 respectively. For *Landsat* and *Peppers* test images, a λ -*Max* operator has been measured for which the marker image has been obtained by subtracting the constant values 20 and 40 respectively to the mask image.

As explained in section 4.3, the most promising parallel algorithms uses two marker image scanings in order to detect initiator pixels and thus, we only analyse in this section the four algorithms based on this technique :

- **algorithm 1** : synchronous propagation step and use of classical FIFO,
- **algorithm 2** : asynchronous propagation step and use of classical FIFO,
- **algorithm 3** : synchronous propagation step and use of hierarchical FIFO,
- **algorithm 4** : asynchronous propagation step and use of hierarchical FIFO.

Figure 8 shows the speedup coefficients obtained by all proposed algorithms by using the four test images. First of all, we can note that these coefficients are limited because of, on one hand, the irregularity of the proposed algorithms, and

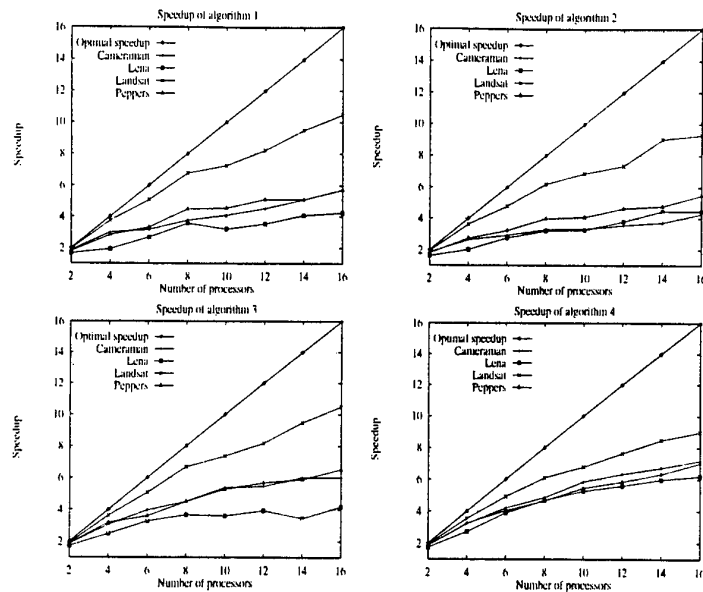


Fig. 8. Speedup coefficients of all proposed algorithms

on the other hand, the small size of the processed images. However, these results well show the behavior of our algorithms. We can thus observe that synchronous approaches (algorithms 1 and 3) are not adapted to the marked irregularity of the algorithms. Indeed, the experimentations have shown that the number of iterations n_{it} needed to reach global stabilization increases with the number of processors. Indeed, as a propagation can only cross an inter-processor frontier at each communication step, it is straightforward that n_{it} is bounded by the number of processors. This fact explains the chaotic behavior of the algorithms 1 and 3 for the *Lena* test image from 10 processors. Moreover, we can remark that these synchronous algorithms can also bring a totally sequential behavior for some images where the evolution of the grey-level values is continuous from top to bottom. As a result, we can conclude that parallel geodesic reconstruction algorithms based on synchronous approach are not scalable.

The algorithm 2 presents very limited speedup caused by the use of a classical FIFO data structure. Indeed, as explained in section 4, this data structure does not ensure the unicity of modification of the pixels and thus, a communication is performed each time a pixel located on an inter-processor frontier is modified. As a result, the number of communication is higher than for synchronous approaches since for these approaches, a communication is performed only after the termination of all local reconstruction.

Finally, the algorithm 4 gives the best results and shows the most regular behavior for all test images. For this small image size, the relative efficiency

is always superior to 40% and we can note that this algorithm shows the best scalability.

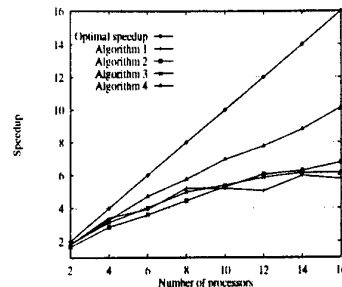


Fig. 9. Speedup of algorithms with a variable problem size

In order to further study the scalability of our algorithms, we have tested them on a variable problem size. For this purpose, we have used test images of size 512×512 . Figure 9 shows speedup coefficients of the four proposed algorithms on the *Lena* test image and for the opening by reconstruction $\rho(\gamma_{15}(f) | f)$. In this Figure, we can note that algorithm 4 shows again the best behavior and the efficiency is here very satisfying since it varies from 63% to 91% whereas the efficiency of all other algorithms is inferior to 38% with 16 processors.

From all of these experimentations, we can conclude that algorithm 4 is the best suited for the implementation of connected operators based on reconstruction process. However, the experimentations have shown a slight load imbalance due to the distribution of grey-level values among processors. Indeed, a processor that reconstructs a dark mask sub-image can receive propagations from neighbor processors, that can generate a large number of re-computation whereas a processor owning a bright sub-image sends a lot of propagations toward its neighbors but receives only a few number of propagations resulting in a shorter execution time.

In order to solve this problem, works are in progress to balance the workload among processors. The retained technique is based on elastic load balancing strategy proposed in [16] and on a progressive reconstruction. First, we split the interval of grey-level values into an arbitrary number of sub-intervals. Then, at the same time, each processor only reconstructs in its own sub-image, the pixels whose grey-level value belong to the same sub-interval. When all pixels of a given sub-interval are reconstructed, a synchronization barrier is called in order to ensure that, at each time, all processors reconstruct pixels with approximatively the same grey-level value. After this call, we change the current sub-interval and a dynamic load balancing scheme is executed to distribute the same grey-level values to all pixels. The remaining problem is to give a method (adaptive or empirical) method to split the interval of grey levels.

6 Conclusion and Perspectives

In this paper, we have presented several parallel algorithms for geodesic reconstruction based on irregular data structures. We have shown that these transformations present complex propagation and re-computation phenomena that have been solved in this paper by using an asynchronous approach as well as a hierarchical data structure. The resulting algorithm shows a marked irregularity but has proved its efficiency for all test images. Currently, works are in progress in order to solve the load imbalance problem.

References

1. Jean Serra. *Image Analysis and Mathematical Morphology*. Academic Press, 1982.
2. Serra J. and Salembier P. Connected Operators and Pyramids. In *Proceedings SPIE Image Algebra and Mathematical Morphology*, volume 2030, pages 65-76, San Diego, 1993.
3. Vincent L. Morphological Grayscale Reconstruction in Image Analysis : Applications and Efficient Algorithms. *IEEE Transactions on Image Processing*, 2(2):176-201, April 1993.
4. Salembier P. and Serra J. Flat Zones Filtering, Connected Operators and Filters by Reconstruction. *IEEE Transactions on Image Processing*, 4(8):1153-1160, August 1995.
5. Crespo J., Serra J. and Schafer R.W. Theoretical Aspects of Morphological Filters by Reconstruction. *Signal Processing*, 47:201-225, 1995.
6. Meyer F., Oliveras A., Salembier P. and Vachier C. Morphological Tools for Segmentation : Connected Filters and Watersheds. *Annales des Télécommunications*, 52(7-8):367-379, July 1997.
7. Salembier P. Morphological Multiscale Segmentation for Image Coding. *Signal Processing*, 38:359-386, 1994.
8. Salembier P. and Pardàs M. Hierarchical Morphological Segmentation for Image Sequence Coding. *IEEE Transactions on Image Processing*, 3(5):639-651, September 1994.
9. Gu C. *Multivalued Morphology and Segmentation-Based Coding*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 1995.
10. Salembier P. and Kunt M. Size-sensitive Multiresolution Decomposition of Images with Rank Order Filters. *Signal Processing*, 27:205-241, 1992.
11. Li W., Haese-Coat V. and Ronsin J. Residues of Morphological Filtering by Reconstruction for Texture Classification. *Pattern Recognition*, 30(7):1081-1093, 1997.
12. Vincent L. *Algorithmes Morphologiques à Base de Files d'Attente et de Lacets. Extension aux Graphes*. PhD thesis, Ecole Nationale Supérieure des Mines de Paris, France, May 1990.
13. Moga A. *Parallel Waterhed Algorithms for Image Segmentation*. PhD thesis. Tampere University of Technology, Finland, February 1997.
14. Breen E. and Monro D.H. An Evaluation of Priority Queues for Mathematical Morphology. In Jean Serra and Pierre Soille, editors, *Mathematical Morphology and Its Applications to Image Processing*, pages 249-256. Kluwer Academic, 1994.
15. Message Passing Interface Forum. *MPI : A Message Passing Interface Standard*, May 1994.
16. Pierson J.M. *Equilibrage de charge dirigé par les données. Applications à la synthèse d'images*. PhD thesis, Ecole normale supérieure de Lyon, October 1996.

Dynamic Load Balancing in Crashworthiness Simulation

Hans Georg Galbas and Otto Kolp

GMD German National Research Center for Information Technology
Institute for Algorithms and Scientific Computing
Schloss Birlinghoven, D-53754 Sankt Augustin, Germany

Abstract. In the numerical simulation of crashworthiness the use of parallel architectures is becoming more and more important. This stems from the desire of engineers in the motorcar industry to get run times which make a dialogue feasible. Parallel computation seems to be the only way to solve these problems in an acceptable time. The computations inherent in crashworthiness simulation can be divided into a contact and a non-contact part. The contact part leads in contrast to the non-contact part to an unbalance due to the uneven (in space and time) distribution of contact. Good scalability becomes a challenge. In this paper we present a dynamic load balancing strategy for crash simulation. It keeps the contact and the non-contact part of the computation separately balanced over the whole simulation time. Results of the dynamic load balancing algorithm are discussed for a contact search algorithm applied to it.

1 Introduction

In the numerical simulation of crashworthiness the use of parallel architectures is becoming more and more important. This stems from the desire of engineers in the motorcar industry to get run times which make a dialogue feasible. Today Finite-Element models for cars, consist of approximately 250000 elements, and more than 100000 time-steps are needed in explicit time-marching schemes. Parallel computation seems to be the only way to solve these problems in an acceptable time. For small parallel systems (less than 16 processors) with shared-memory the obtainable speedups are very satisfactory. But this performance decreases significantly with increasing processor numbers. So for large problems of crash simulation distributed-memory MIMD architectures are the better choice. Standard domain partitioning tools (see [1] and [2]) employing a recursive spectral bisection algorithm and trying to find connected parts of equal size for each processor, generate partitions with a good balanced workload for the finite element part of the crash simulation algorithm. In general however the contact-impact part of the computation is distributed very unevenly (in space and time) by such a partition which leads to an undesired load unbalance. Good scalability becomes a challenge for crash simulation. Some improvements were obtained by giving higher weights (in the partitioning tool) to elements with expected contact and making blocks of the partition with high contact smaller [3].

To develop efficient parallel contact search algorithms the KALCRASH Project, funded by the German Ministry for Research and Education (BMBF), was carried out. The research led to a remarkable improvement concerning the contact calculations [4]. This was valid for the sequential as well as the parallel case. A part of the performance improvements was obtained by a static load balancing strategy [5].

2 From Static to Dynamic Load Balancing

In the following discussion we differ between two parts of the program for crash simulation, the contact part (CP) and the non-contact part (NCP). One observation is the fact that the two parts are connected by necessary communications. The synchronizing effects of communication between CP and NCP in each simulation step make the strategy to compensate load unbalance in one part by a suitable load unbalance of the other part not very successful. Apart from communication the overall computation time is determined mainly by the sum of the maximal computation times for each part over all processors.

These observations led to the static load balancing strategy to distribute similar workloads to each processor for each part (CP and NCP). Instead of using the partition of the domain partitioning tool directly which optimises the workload for NCP, a refined partition i.e. for a multiple number of processors, was considered (over-partitioning). Then blocks of the refined partition with high workload concerning CP are combined with blocks with low workload concerning CP to form a block of a new partition for the given number of processors. The blocks of the refined partition are now subblocks of the new partition. This new partition fulfils the condition of similar workloads for all processors for each part (CP and NCP). For example a block of the refined partition, (of the crashing zone i.e. front of the car) was combined with another block of the refined partition of the rear of the car to build a block of a new partition.

A drawback of this approach is that a block generally consists of different separated subblocks which leads to some overhead in the finite element part of crash simulation concerning communication. This effect is investigated in the running EU Project SIM-VR where the discussed contact search algorithm (CSA) was successfully integrated into the crash simulation code PAM-CRASH of ESI. First results show that the gain in the contact computations by far outweighs the loss in the non-contact part.

Another drawback of the static load balancing approach is that some skill and time is needed to put the right subblocks together to obtain a good partition. A good partition is a partition which fulfils the condition of similar low workloads for all processors for CP and NCP in the average concerning the whole run time. Since the workload concerning CP is changing a dynamic load balancing strategy based on the static approach is to perform an exchange of subblocks from time to time automatically. Once a good partition is found it remains for some time a fairly good partition since the model itself changes in general only slowly (with respect to time-steps). This dynamic load balancing approach lies

with regard to its granularity to repartition every time-step and using connected and compact blocks [6] on the other extreme. This work and the integration of CSA with dynamic load balancing into PAM-CRASH is now to be performed in the BMBF-Project AUTOBENCH.

3 Dynamic Load Balancing by Over-Partitioning

Starting with an arbitrary refined partition (the number of blocks is a multiple of the number of processors) given by a domain partitioning tool each processor is associated with the same number of subblocks. This 'guaranties' a good load balance for the non-contact part. Since the time-steps in the simulation are largely synchronized by the communication structure and by construction we get a good load balance for the non-contact part it remains to achieve good load balance in the contact part. This is done by an exchange of subblocks.

3.1 When to make Load Balancing Steps

With a given partition the simulation is performed for a certain number (nstep, see Table 1) of time-steps but in accordance with the multi-level contact search algorithm [5] (i.e. outside the fast loop). At this event each processor measures the time it needs for certain tasks which are representative for the workload of the contact part. These tasks include the creation of lists of neighbouring but nonconnected nodes for each slave node which is admissible for contact and the measuring of distances of elements associated with these nodes to the slave nodes. These times are made available in an all-to-all communication to all processors. If these times differ by no more than the special threshold parameter exstop (see Table 1) no load balancing is performed and the simulation is continued for the next nstep time-steps after which the same procedure is repeated. If they differ by more than exstop a load balancing step is initiated.

3.2 How to make Load Balancing Steps

To combine the right subblocks into one block the workload for the contact part of each subblock has to be determined. Since these subblocks are not treated separately in CSA the workload caused by them can not be measured directly. It is measured indirectly through the number of nodes and proximity-pairs (a slave node and a nearby nonconnected element) which have been detected by CSA inside the corresponding subblocks. A good estimation model for the workload was achieved by a regression of the time spent for certain contact search computations inside a block on the size of these sets of nodes and proximity-pairs. These sets represent the hierarchical strategy of CSA: elimination of nodes and elements from current contact search as early and cheaply as possible in order to minimize computation and communication.

The estimated workload of contact computation for a processor is given by the sum of the estimated workloads of the subblocks which form a block. Improvement is assumed if through a recombination of subblocks the maximal estimated workload over all processors can be reduced. If improvement is possible the corresponding processors exchange suitable subblocks. A typical block for an eight processor run formed by a recursive spectral bisectioning tool is shown by the dark region in Fig. 1. The block is connected and compact. All of it lies in a zone with likely contact. This induces load unbalance in the contact part of the simulation. After an exchange of subblocks the block associated with the same



Fig. 1. Block of a BMW model from standard partitioning for 8 processors.

processor has changed into one with two connectivity components (see Fig. 2). The part in the front of the car where the most deformation takes place has intensive contact whereas the part in the rear of the car has almost no contact. The new block has inclusive communication less workload for the contact part of the simulation than the old one.

The new algorithm performs this operation automatically at certain instances (when the observed load unbalance exceeds the given threshold parameter *exstop*) to minimize the maximal workload over all processors. This operation introduces of course additional overhead but this is controlled sufficiently by the above-mentioned threshold parameters *exstop* and *nstop*.

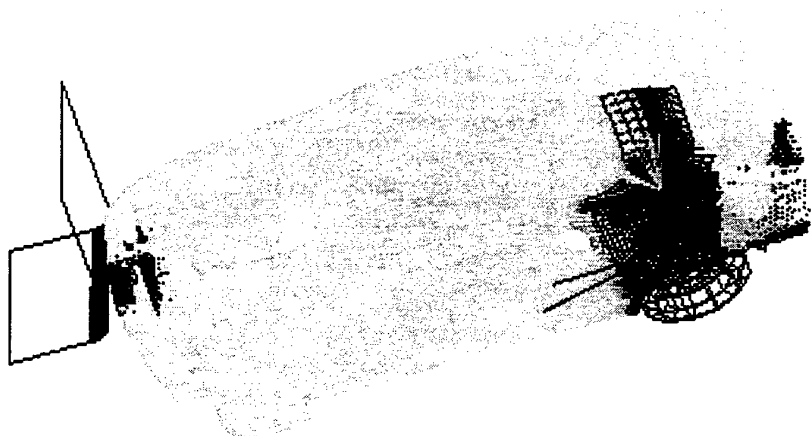


Fig. 2. Block of a BMW model from over-partitioning for 8 processors.

4 Performance Results

First performance results of the dynamic load balancing strategy have been obtained for the contact search program CSA. CSA is implemented in Fortran77 with the message-passing interface MPI. Since CSA with dynamic load balancing is not yet implemented into PAM-CRASH we use interpolated data as in [5]. They are based on a 40% off-set crash simulation of a BMW benchmark model with around 60000 elements.

Simulations of 40000 time-steps on an IBM SP2 are considered for 8, 16 and 32 processors. The dynamic load balancing partitions are created from a fourfold over-partitioning, i.e. each processor has got four subblocks (see Table 1). There are two 'start' partitions for the dynamic case. The first are the ones given by the partitioning tool directly (the four subblocks created through bisection in the last two steps of the partitioning tool are combined to one block) and the second are the static load balancing partitions created from a fourfold over-partitioning which were found to be very good in the static load balancing approach [5]. Comparing the run times for contact search without dynamic load balancing for this two partitions (labeled 'without' and 'static' in Table 1) we see that in case of 8 processors the time is halved and in case of 16 resp. 32 processors an improvement of 36% resp. 40% was obtained. The improvement is measured against run times with no load balancing.

Considering now the dynamic load balancing results we have varied the parameters *nstep* (number of time-steps until the next workload monitoring) and *exstop* (maximal difference in measured workload over all processors). The dynamic load balancing starts from the partition without load balancing and from

Table 1. Dynamic load balancing results for 40000 time-steps with fourfold over-partitioning.

number of proc	number of subblocks	load balancing	nstep	exstop (%)	time (sec)	improvement (%)
8	8	without			2615	0
8	32	static			1269	51
8	32	dynamic	500	10	1445	45
8	32	dynamic	1000	5	1449	45
8	32	dynamic	1000	10	1309	50
8	32	dynamic	1000	15	1375	47
8	32	dynamic	2000	10	1386	47
8	32	stat+dyn	1000	5	1347	48
8	32	stat+dyn	1000	10	1295	50
8	32	stat+dyn	1000	15	1321	49
16	16	without			1829	0
16	64	static			1176	36
16	64	dynamic	500	15	958	48
16	64	dynamic	1000	10	949	48
16	64	dynamic	1000	15	941	48
16	64	dynamic	1000	20	933	49
16	64	dynamic	2000	15	945	48
16	64	stat+dyn	1000	10	929	49
16	64	stat+dyn	1000	15	903	51
16	64	stat+dyn	2000	15	888	51
16	64	stat+dyn	2000	15	874	52
32	32	without			1134	0
32	128	static			685	40
32	128	dynamic	500	15	843	26
32	128	dynamic	1000	10	824	28
32	128	dynamic	1000	15	778	31
32	128	dynamic	1000	20	784	31
32	128	dynamic	2000	15	776	32
32	128	stat+dyn	1000	10	750	34
32	128	stat+dyn	1000	15	725	36
32	128	stat+dyn	1000	20	768	32
32	128	stat+dyn	2000	15	721	36

the static load balancing partitions ('stat+dyn', see Table 1). The computing times for the latter case are slightly better because of the better starting partition. For example for 32 processors the improvement rose from 32% to 36%. It can be seen that the times for dynamic load balancing are similar to the times in the static load balancing case. In case of 16 processors the performance was better. The exchanges of subblocks become less frequent while the simulation is going on. Altogether between 50 and 100 exchanges within the first 40000 time-steps were observed for the dynamic cases with 32 processors.

The two parameters *nstep* and *exstop* have to be adapted to the crash model. They show that one should not try too few or too many times to find a better partition and that the difference in workload between the processors must be worthwhile. In Table 2 the partitions are created from two- or eight-fold over-partitioning. While the twofold over-partitioning leads to big subblocks in the front with intensive contact the eight-fold over-partitioning leads to complex blocks with more communication (more neighbours) (see Table 2) but better adjustments. Since the eight-fold over-partitioning implies the fourfold case (the refined partition was created by a bisectioning procedure) a more elaborate designed objective function incorporating communication requirements should lead to some further improvement. It seems that decreasing the granularity further leads to more (untractable) overhead (see Table 2, 8 processor case with 128 subblocks).

Table 2. Dynamic load balancing results for 40000 time-steps with two-, eight- and sixteen-fold over-partitioning.

number of proc	number of subblocks	load balancing	nstep	exstop (%)	time (sec)	improvement (%)
8	16	dynamic	1000	15	2004	23
8	32	dynamic	1000	15	1375	47
8	64	dynamic	1000	15	1275	51
8	128	dynamic	1000	15	1428	45
16	32	dynamic	1000	15	1306	29
16	64	dynamic	1000	15	941	48
16	128	dynamic	1000	15	900	51
32	64	dynamic	1000	15	899	21
32	128	dynamic	1000	15	778	31

5 Concluding Remarks

Future work especially the integration of CSA with dynamic load balancing into PAM-CRASH will be done in the project AUTOBENCH. One topic will be the integration of communication into the objective function for determining

the subblocks to exchange. To find good partitions the number of neighbouring blocks in a partition and the number of boundary elements of these blocks will also be considered. So for instance the rule that neighbours in the front should also be neighbours in the rear seems to be a good strategy since it leads in tendency to less communication.

Acknowledgements

The authors would like to thank their colleagues in GMD-SCAI for their support.

References

1. Floros, N., Reeve, J.S., Clinckemallie, J., Vlachoutsis, S., Lonsdale, G.: Practical aspects and experiences, Comparative efficiencies of domain decompositions. *Parallel Computing* **21** Elsevier Science B.V., 1995.
2. Persson, P.: *Parallel Numerical Procedures for the Solution of Contact-Impact Problems*. Linköping Studies in Science and Technology, Thesis No. 584, UniTryck, Linköping 1996.
3. Clinckemallie, J., Elsner, B., Lonsdale, G., Meliciani, S., Vlachoutsis, S., de Bruyne, F., Holzner, M.: Performance Issues of the Parallel PAM-CRASH Code. *The International Journal of Supercomputer Applications and High Performance Computing*, Vol. 11, No. 1, Sage Publications, Inc., 1997.
4. Elsner, B., Galbas, H.G., Görg, B., Kolp, O., Lonsdale, G.: A Parallel, Multi-level Approach for Contact Problems in Crashworthiness Simulation. *Parallel Computing: State-of-the-Art and Perspectives*, Proceedings of the Int. Conf. Parco95, Elsevier Science B.V., 1996.
5. Elsner, B., Galbas, H.G., Görg, B., Kolp, O., Lonsdale, G.: A Parallel, Multi-level Contact Search Algorithm in Crashworthiness Simulation. *Advances in Computational Structures Technology*, Civil-Comp Press, Edinburgh, 1996.
6. Attaway, S., Hendrickson, B., Swegle, J., Vaughan, C., Gardner, D.: Transient Dynamics Simulations: Parallel Algorithms for Contact Detection and Smoothed Particle Hydrodynamics. *Supercomputing 96 Technical Papers*, Pittsburgh, 1996.

A Parallelization Strategy for Power Systems Composite Reliability Evaluation

Carmen L.T. Borges

Djalma M. Falcão

Federal University of Rio de Janeiro
C.P. 68504, 21945-970, Rio de Janeiro - RJ
Brazil

carmen@coep.ufrj.br, falcão@coep.ufrj.br

Abstract. This paper presents a methodology for parallelization of the power systems composite reliability evaluation using Monte Carlo simulation. A coarse grain parallelization strategy is adopted, where the adequacy analyses of the sampled system states are distributed among the processors. An asynchronous parallel algorithm is described and tested on five different electric systems. The paper presents results of almost linear speedup and high efficiency obtained on a four nodes IBM RS/6000 SP distributed memory parallel computer.

1 Introduction

The power generation, transmission and distribution systems constitute a basic element in the economic and social development of the modern societies. For technical and economic reasons, these systems have evolved from a group of small and isolated systems to large and complex interconnected systems with national or, even, continental dimensions. For this reason, failure of certain components of the system can produce disturbances capable of affecting a great number of consumers. On the other hand, due to the sophistication of the electric and electronic equipments used by the consumers, the demand in terms of the power supply reliability has been increasing considerably. More recently, institutional changes in the electric energy sector, such as those originated by deregulation policies, privatizations, environmental restrictions, etc., have been forcing the operation of such systems closer to its limits, increasing the need to evaluate the power supply interruption risks and quality degradation in a more precisely form.

Probabilistics models have been largely used in the evaluation of the power systems performance. Based on information about components failures, these models allow to establish system performance indexes which can be used to aid decision making relative to new investments, operative policies and to evaluate transactions in the electric energy market. This type of study receives the generic name of Reliability Evaluation [1] and can be accomplished in the generation, transmission and distribution levels or, still, combining the several levels. The composite system reliability evaluation refers to the reliability evaluation of

power systems composed of generation and transmission sub-systems, object of this work.

The basic objective of the composite generation and transmission system reliability evaluation is to assess the capacity of the system to satisfy the power demand at its main points of consumption. For this purpose, it is considered the possibility of occurrence of failures in both generation and transmission system components and the impact of these failures in the power supply is evaluated. There are two possible approaches for reliability evaluation: analytic techniques and stochastic simulation (Monte Carlo simulation [2, 3], for example). In the case of large systems, with complex operating conditions and a high number of severe events, the Monte Carlo simulation is, generally, preferable due to the easiness of modeling complex phenomena.

For the reliability evaluation based on Monte Carlo simulation, it is necessary to analyze a very large number of system operating states. This analysis, in general, includes load flow calculations, static contingencies analysis, generation re-scheduling, load shedding, etc. In several cases, this analysis must be repeated for several different load levels and network topology scenarios.

The reliability evaluation of large power systems may demand hours of computation on high performance workstations. The majority of the computational effort is concentrated in the system states analysis phase. This analysis may be performed independently for each system state. The combination of elevated processing requirements with the concurrent events characteristic suggests the application of parallel processing for the reduction of the total computation time.

This paper describes some results obtained through a methodology under development for power system composite reliability evaluation on parallel computers with distributed memory architecture and communication via message passing. The methodology is being developed using as reference element a sequential program for reliability evaluation used by the Brazilian electric energy utilities [4].

2 Composite Reliability Evaluation

The composite generation and transmission system reliability evaluation consists of the calculation of several performance indexes, such as the Lost of Load Probability (LOLP), Expected Power Not Supplied (EPNS), Lost of Load Frequency (LOLF), etc., using a stochastic model for the electric system operation. The conceptual algorithm for this evaluation is as follows:

1. *Select an operating scenario \underline{x} corresponding to a load level, components availability, operating conditions, etc.*
2. *Calculate the value of an evaluation function $F(\underline{x})$ which quantifies the effect of violations in the operating limits in this specific scenario. Corrective actions such as generation re-scheduling, load shedding minimization, etc., can be included in this evaluation.*
3. *Update the expected value of the reliability indexes based on the result obtained in step 2.*

4. If the accuracy of the estimates is acceptable, terminate the process. Otherwise, return to step 1.

Consider a system with n components (transmission lines, transformers, electric loads, etc.). An operating scenario for this system is given by the random vector:

$$\underline{x} = (x_1, x_2, \dots, x_m) , \quad (1)$$

where x_i is the state of the i -th component. The group of all possible operating states, obtained by all the possible combinations of components states, is called state space and represented by X . In Monte Carlo simulation, step 1 of the previous algorithm consists of obtaining a sample of vector $\underline{x} \in X$, by sampling the random variables probability distributions corresponding to the components operating states, using a random number generator algorithm.

In step 2 of the previous algorithm, it is necessary to simulate the operating condition of the system in the respective sampled states, in order to determine if the demand can be satisfied without operation restrictions violation. This simulation demands the solution of a contingency analysis problem [5] and, eventually, of an optimal load flow problem [6] to simulate the generation re-scheduling and the minimum load shedding. In the case of large electric systems, these simulations require high computational effort in relation to that necessary for the other steps of the algorithm [7].

The reliability indexes calculated at step 3 correspond to estimates of the expectation of different evaluation functions $F(\underline{x})$, obtained for N system state samples by:

$$\bar{E}[F] = \frac{1}{N} \sum_{k=1}^N F(\underline{x}_k) . \quad (2)$$

The Monte Carlo simulation accuracy may be expressed by the coefficient of variation α , which is a measure of the uncertainty around the estimate, and defined as:

$$\alpha = \frac{\sqrt{V(\bar{E}[F])}}{\bar{E}[F]} . \quad (3)$$

The convergence criterion usually adopted is the coefficient of variation of the EPNS index, which has the worst convergence rate of all reliability indexes.

3 Parallelization Strategy

The composite reliability evaluation problem can be summarized in three main functions: the system states selection, the adequacy analysis of the selected system states and the reliability indexes calculation. As described before, an adequacy analysis is performed for each state selected by sampling in the Monte

Carlo simulation, i.e., the system capacity to satisfy the demand without violating operation and security limits is verified.

The algorithm is inherently parallel with a high degree of task decoupling [8]. The system states analyses are completely independent from each other and, in a coarse grain parallelization strategy, it is only necessary to communicate at three different situations:

1. For the initial distribution of data, identical for all processors and executed once during the whole computation;
2. For the final grouping of the partial results calculated in each processor, also executed only once ; and
3. For control of the global parallel convergence, which needs to be executed several times during the simulation process, with frequency that obeys some convergence control criteria.

The basic configuration for parallelization of this problem is the master-slaves paradigm, in which a process, denominated master, is responsible for acquiring the data, distributing them to the slaves, controlling the global convergence, receiving the partial results from each slave, calculating the reliability indexes and generating reports. The slaves processes are responsible for analyzing the system states allocated to them, sending their local convergence data to the master and, at the end of the iterative process, also sending their partial results. It is important to point out that, in architectures where the processors have equivalent processing capacities, the master process should also analyze system states in order to improve the algorithm performance. For purposes of this work, each process is allocated to a different processor and is referred to simply as processor from now on.

The main points that had to be solved in the algorithm parallelization were the system states distribution philosophy and the parallel convergence control criteria, in order to have a good load balancing. These questions are dealt with in the next subsections.

3.1 System States Distribution Philosophy

The most important problem to be treated in parallel implementations of Monte Carlo simulation is to avoid the existence of correlation between the sequences of random numbers generated in different processors. If the sequences are correlated in some way, the information produced by different processors will be redundant and they will not contribute to increase the statistical accuracy of the computation, degrading the algorithm performance. In some Monte Carlo applications, correlation introduces interferences that produce incorrect results. To initialize the random numbers generator with different seeds for each processor is not a good practice, because it can generate sequences that are correlated to each other [9].

In the system states distribution philosophy adopted, the system states are generated directly at the processors in which they will be analyzed. For this purpose, all processors receive the same seed and execute the same random numbers

sampling, generating the same system states. Each processor, however, starts to analyze the state with a number equal to its rank in the parallel computation and analyzes the next states using as step the number of processors involved in the computation. Supposing that the number of available processors is 4, then processor 1 analyzes states numbered 1,5,9,..., processor 2 analyzes states numbered 2,6,10,..., and so on.

3.2 Parallel Convergence Control

Three different parallelization strategies for the composite reliability evaluation problem were tried [10] with variations over the task allocation graph, the load distribution criteria and the convergence control method. The strategy that produces best results in terms of speedup and scalability is an asynchronous one that will be described in details in this section. The task allocation graph for this asynchronous parallel strategy is shown in Fig. 1, where p is the number of scheduled processors. Each processor has a rank in the parallel computation which varies from 0 to $(p-1)$, 0 referring to the master process. The basic tasks involved in the solution of this problem can be classified in five types: **I** - Initialization, **A** - States Analysis, **C** - Convergence Control, **P** - Iterative Process Termination and **F** - Finalization (calculation of reliability indexes and generation of reports). A subindex i associated with task T means it is allocated to processor i . A superindex j associated with T_i means the j -th execution of task T by processor i .

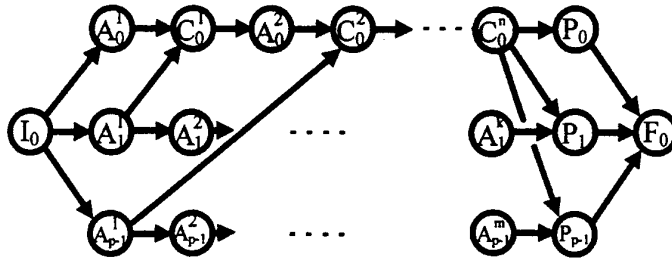


Fig. 1. Task Allocation Graph

The problem initialization, which consists of the data acquisition and some initial computation, is executed by the master processor, followed by a broadcast of the data to all slaves. All processors, including the master, pass to the phase of states analysis, each one analyzing different states. After a time interval Δt , each slave sends to the master the data relative to its own local convergence, independently of how many states it has analyzed so far, and then continues to

analyze other states. At the end of another Δt time interval, a new message is sent to the master, and this process is periodically repeated.

When the master receives a message from a slave, it verifies the status of the global parallel convergence. If it has not been achieved, the master goes back to the system states analysis task until a new message arrives and the parallel convergence needs to be checked again. When the parallel convergence is detected or the maximum number of state samples is reached, the master broadcasts a message telling the slaves to terminate the iterative process, upon what the slaves send back their partial results to the master. The master then calculates the reliability indexes, generate reports and terminate.

In this parallelization strategy, there is no kind of synchronization during the iterative process and the load balancing is established by the processors capacities and the system states complexity. The precedence graph [11] is shown in Fig. 2, where the horizontal lines are the local time axis of the processors. In this figure only the master and one slave are represented. The horizontal arches represent the successive states by which a processor passes. The vertices represent the messages sending and receiving events. The messages are represented by the arches linking horizontal lines associated with different processors.

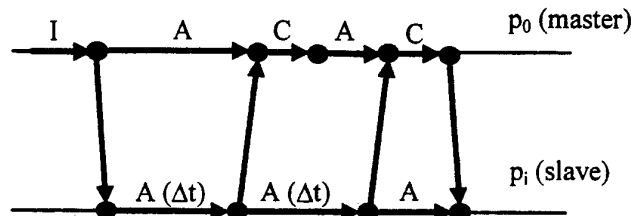


Fig. 2. Precedence Graph

An additional consideration introduced by the parallelization strategy is the redundant simulation. During the last stage of the iterative process, the slaves execute some analyses that are beyond the minimum necessary to reach convergence. The convergence is detected based on the information contained in the last message sent by the slaves and, between the shipping of the last message and the reception of the message informing of the convergence, the slaves keep analyzing more states. This, however, does not imply in loss of time for the computation as a whole, since no processor gets idle any time. The redundant simulation is used for the final calculation of the reliability indexes, generating indexes still more accurate than the sequential solution, since in Monte Carlo methods the uncertainty of the estimate is inversely proportional to the number of analyzed states samples.

4 Implementation

4.1 Message Passing Interface (MPI)

MPI is a standard and portable message passing system designed to operate on a wide variety of parallel computers [12]. The standard defines the syntax and semantics of the group of subroutines that integrate the library. The main goals of MPI are portability and efficiency. Several efficient implementations of MPI already exist for different computer architectures. The MPI implementation used in this work is the one developed by IBM for AIX that complies with MPI standard version 1.1.

Message passing is a programming paradigm broadly used in parallel computers, especially scalable parallel computers with distributed memory and networks of workstations (NOWs). The basic communication mechanism is the transmittal of data between a pair of processes, one sending and the other receiving. There are two types of communication functions in MPI: blocking and non-blocking. In this work, the non-blocking send and receive functions are used, what allow the possible overlap of message transmittal with computation and tend to improve the performance.

Other important aspects of communication are related to the semantics of the communication primitives and the underlying protocols that implement them. MPI offers four modes for point to point communication which allow to choose the semantics of the send operation and to influence the protocol of data transferring. In this work, the Standard mode is used, in which it is up to MPI to decide whether outgoing messages are buffered or not based on buffer space availability and performance reasons.

4.2 Parallel Computer

The IBM RS/6000 SP Scalable POWERparallel System [13] is a scalable parallel computer with distributed memory. Each node of this parallel machine is a complete workstation with its own CPU, memory, hard disk and net interface. The architecture of the processors may be POWER2/PowerPC 604 or Symmetrical Processor (SMP) PowerPC. The nodes are interconnected by a high performance switch dedicated exclusively for the execution of parallel programs. This switch can establish direct connection between any pair of nodes and one full-duplex connection can exist simultaneously for each node.

The parallel platform where this work was implemented is an IBM RS/6000 SP with 4 POWER2 processors interconnected by a high performance switch of 40 MB/s full-duplex bandwidth. Although the peak performance for floating point operations is 266 MFLOPS for all nodes of this machine, there are differences related to processor type, memory and processor buses, cache and RAM memory that can be more or less significant depending on the characteristics of the program in execution.

5 Results

5.1 Test Systems

Five different electric systems were used to verify the performance and scalability of the parallel implementation. The first one is the IEEE-RTS standard reliability test system for reliability evaluation [14]. The second one, CIGRÉ-NBS is a representation of the New Brunswick Power System proposed by CIGRÉ as a standard for reliability evaluation methodology comparison [15]. The third, fourth and fifth ones are representations of the actual Brazilian power system, with actual electric characteristics and dimensions, for region North-Northeastern (NNE), Southern (SUL) and Southeastern (SE), respectively. The main data for the test systems are shown in Table 1. It was adopted a convergence tolerance of 5% in the EPNS index for all test systems reliability evaluation.

Table 1. Test Systems

System	Nodes	Circuits	Areas
RTS	24	38	2
NBS	89	126	4
NNE	89	170	6
SUL	660	1072	18
SE	1389	2295	48

5.2 Results Analysis

The speedups and efficiencies achieved by the parallel code for the five test systems on 2, 3 and 4 processors, together with the CPU time of the sequential code (1 processor), are summarized in Table 2.

Table 2. Results

System	CPU time	Speedup			Efficiency		
	p=1	p=2	p=3	p=4	p=2	p=3	p=4
RTS	35.03 sec	1.87	2.70	3.52	93.68	90.07	88.07
NBS	24.36 min	1.96	2.94	3.89	98.10	97.90	97.28
NNE	17.14 min	1.99	2.99	3.98	99.75	99.63	99.44
SUL	25.00 min	1.95	2.91	3.85	97.71	97.00	96.28
SE	8.52 hour	1.98	2.93	3.88	99.12	97.70	97.04

As it can be seen, the results obtained are very good, with speedup almost linear and efficiency close to 100 % for the larger systems. The asynchronous

parallel implementation provides practically ideal load balancing and negligible synchronization time. The communication time for broadcasting of the initial data and grouping of the partial results is negligible compared to the total time of the simulation. The communication time that has significant effect in the parallel performance is the time consumed in exchanging messages for controlling the parallel convergence. The time spent by the processors sampling states that are not analyzed by them, due to the states distribution philosophy adopted, is also negligible compared to the total computation time.

The algorithm also presented a good scalability in terms of number of processors and dimension of the test systems, with almost constant efficiency for different numbers of processors. This good behavior of the parallel solution is due to the combination of three main aspects:

1. The high degree of parallelism inherent to the problem,
2. The coarse grain parallelization strategy adopted and
3. The asynchronous implementation developed.

Figure 3 shows the speedup curve for the RTS and NBS test systems and Fig. 4 for the three actual Brazilian systems.

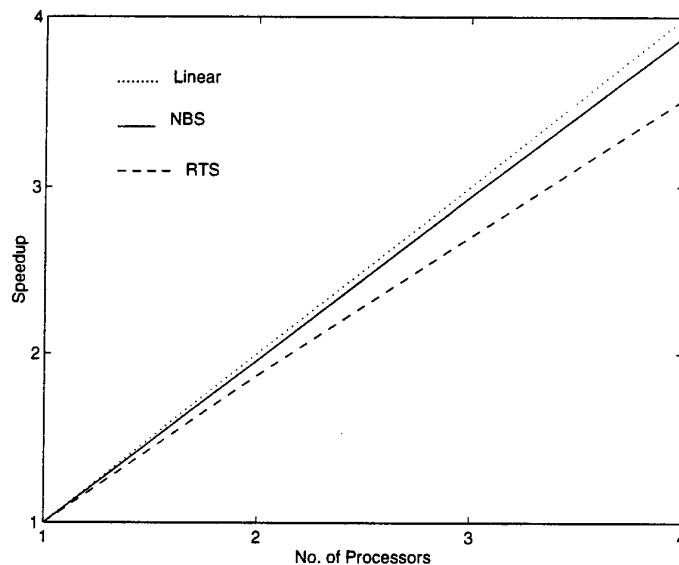


Fig. 3. Speedups - RTS and NBS

A comparison of the main indexes calculated in both sequential and parallel (4 processors) implementations can be done based on Table 3, where the

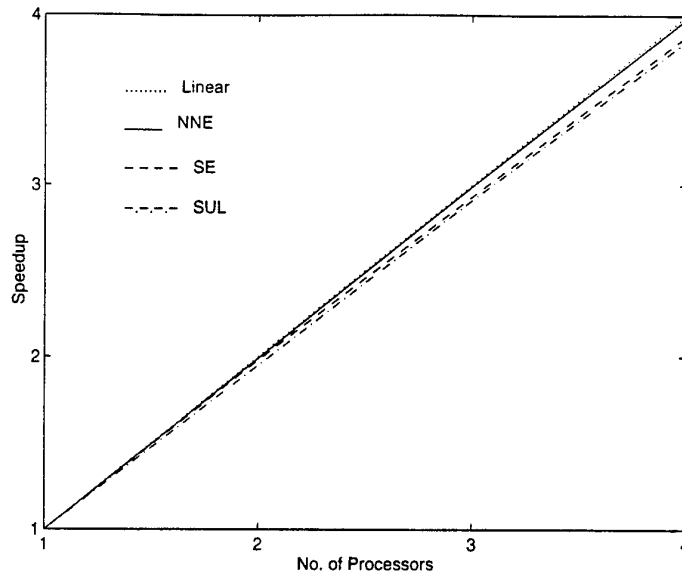


Fig. 4. Speedups - NNE, SUL and SE

Expected Power Not Supplied (EPNS) is given in MW and the Lost of Load Frequency (LOLF) is given in occurrences/year.

Table 3. Reliability Indexes

System	LOLP		EPNS		LOLF		$\alpha(\%)$	
	p=1	p=4	p=1	p=4	p=1	p=4	p=1	p=4
RTS	0.144	0.142	25.16	24.98	26.90	26.28	4.998	4.883
NBS	0.014	0.014	2.10	2.10	10.89	10.88	4.999	4.999
NNE	0.027	0.027	1.99	2.01	23.97	24.23	5.000	4.976
SUL	0.188	0.188	3.82	3.82	69.76	69.61	4.994	4.964
SE	0.037	0.037	1.75	1.75	114.22	114.56	5.000	4.998

As it can be seen, the results obtained are statistically equivalent, with the parallel results slightly more accurate (smaller coefficient of variation α). This is due to the convergence detection criteria. In the sequential implementation, the convergence is checked at each new state analyzed. In the parallel implementation, there is no sense in doing this, since it would imply in a very large number of messages between processors. The parallel convergence is checked in chunks of states analyzed at different processors, as described before, what may lead to a different total of analyzed states when the convergence is detected.

Another factor that contributes to the greater accuracy of the parallel solution is the redundant simulation described earlier.

Another interesting observation is that the convergence path for the parallel implementation is not necessarily the same as the one for the sequential implementation. The states adequacy analyses may demand different computational times depending on the complexity of the states being analyzed. Besides, faster processors may analyze more states than slower ones in the same time interval. The combination of these two characteristics may lead to a different sequence of states analyzed by the overall simulation process, resulting in a different convergence path from the sequential code, but producing practically the same results.

6 Conclusions

The power systems composite reliability evaluation using Monte Carlo simulation demands high computation effort due to the large number of states that need to be analyzed and to the complexity of these states analyses.

Since the adequacy analysis of the system states can be performed independently from each other, the use of parallel processing is a powerful tool for the reduction of the total computation time.

This paper presented a composite reliability evaluation methodology for a distributed memory parallel processing environment. A coarse grain parallelism was adopted, where the processing grain is composed by the adequacy analysis of several states. In this methodology, the states to be analyzed are generated directly at the different processors, using a distribution algorithm that is based on the rank of the processor in the parallel computation.

A parallel implementation was developed where the Monte Carlo simulation convergence is controlled in a totally asynchronous way. This asynchronous implementation has a practically ideal load balancing and worthless synchronization time.

The results obtained in a 4 nodes IBM RS/6000 SP parallel computer for five electric systems are very good, with practically linear speedup, close to the theoretical, and efficiencies also close to 100%. The reliability indexes calculated in both the sequential and parallel implementations are statistically equivalent.

A point being explored in continuation to this work is the migration of the methodology developed from a distributed memory parallel computer to a network of workstations. To ally the computation time reduction achieved by this parallel methodology with the use of networks of workstations already available at the electric energy companies is of great interest from the economic point of view.

References

1. R. Billinton and R.N. Allan, "*Reliability Assessment of Large Electric Power Systems*", Kluwer, Boston, 1988.

2. R. Billinton and W. Li, "Reliability Assessment of Electric Power Systems Using Monte Carlo Methods", Plenum Press, New York, 1994.
3. M.V.F. Pereira and N.J. Balu, "Composite Generation/Transmission System Reliability Evaluation", *Proceedings of IEEE*, vol. 80, no. 4, pp. 470-491, April 1992.
4. "NH2 Computational System for Generation and Transmission Systems Reliability Evaluation - Methodology Manual", (in Portuguese), CEPEL Report no. DPP/POL-137/93, 1993.
5. B. Stott, O. Alsac, A.J. Monticelli, "Security Analysis and Optimization", *Proceedings of IEEE*, Vol. 75, no. 12, pp. 1623-1644, Dec 1987.
6. H.W. Dommel, W.F. Tinney, "Optimum Power Flow Solutions", *IEEE Transactions on Power Apparatus and Systems*, pp. 1866-1876, Oct 1968.
7. A.C.G. Melo, J.C.O. Mello, S.P. Romero, G.C. Oliveira, R.N. Fontoura, "Probabilistic Evaluation of the Brazilian System Performance", *Proceedings of the IV Simpósio de Especialistas em Planejamento da Operação e Expansão Elétrica*, (in Portuguese), 1994.
8. N. Gubbala, C. Singh, "Models and Considerations for Parallel Implementation of Monte Carlo Simulation Methods for Power System Reliability Evaluation", *IEEE Transactions on Power Systems*, Vol. 10, no. 2, pp. 779-787, May 1995.
9. G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, D. Walker, "Solving Problems on Concurrent Processors", Vol.1, Prentice Hall, New Jersey, 1988.
10. C.L.T. Borges, D.M. Falcão, J.C.O. Mello, A.C.G. Melo, "Reliability Evaluation of Electric Energy Systems composed of Generation and Transmission in Parallel Computers", *Proceedings of the IX Simpósio Brasileiro de Arquitetura de Computadores / Processamento de Alto Desempenho*, (in Portuguese), pp. 211-223, Oct 1997.
11. V.C. Barbosa, "An Introduction to Distributed Algorithms", The MIT Press, Cambridge, Massachusetts, 1996.
12. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, "MPI: The Complete Reference", The MIT Press, Cambridge, Massachusetts, 1996.
13. "IBM POWERparallel Technology Briefing: Interconnection Technologies for High-Performance Computing (RS/6000 SP)", RS/6000 Division, IBM Corporation, 1997.
14. IEEE APM Subcommittee, "IEEE Reliability Test System", *IEEE Transaction on Power Apparatus and Systems*, Vol. PAS-98, pp. 2047-2054, Nov/Dec 1979.
15. CIGRÉ Task Force 38-03-10, "Power System Reliability Analysis - Volume 2 - Composite Power Reliability Evaluation", 1992.

Parallel Paradigms applied in a Fluid-Dynamic Problem to Model a Glass Manufacturing Process¹

J. Vinuesa, R. Menéndez de Llano, V. Puente, B. Torón

Grupo de Arquitectura y Tecnología de Computadores
Departamento de Electrónica y Computadores
E.T.S.I.I.T.

Universidad de Cantabria
Avda. de Los Castros, s/n
39005 Santander, Spain

e-mail: {vinuesa, rafa, vpuente, borja}@atc.unican.es

Abstract. Two parallel codes have been designed with the aim of solving a fluid-dynamic problem that appears in a real industrial-scale glass manufacturing process. To solve this problem, the ADI method has been implemented with the two parallel programming paradigms. Taking into account the comparison of programming effort versus portability, the two solutions offer different advantages. In this paper, we also evaluate the high communication costs exhibited by the inherent parallelism in the ADI method.

1. Introduction

This paper focuses on a R+D project developed with the aim of obtaining a parallel code that solves the problems that arise when designing new industrial glass manufacturing plants. Up to now, this type of industrial problem has not been affordable for obtaining a successful computer-based tool in order to model such processes.

The organisation of this paper is as follows: this section describes the modelling of the physical process and the mathematical approach. Section 2 introduces the parallel strategies applied to the parallel machines that have been tested with our benchmark. Section 3 shows the most relevant performance results and explains the different behaviours of the application. Finally, Section 4 concludes with future work that must follow our preliminary results.

¹ This work has been supported and co-funded by the European Commission within the ESPRIT IV, (Project Ref. 21037, HPCN Initiative - PCI II) and the Spanish CICYT under contracts TIC95-0378 and TIC97-1432-CE.

1.1. The Physical Process

The actual glass manufacturing process can be described as follows. In the production plant, there is a closed vat about 60x8 metres that contains a thin layer of liquid tin. In the head of this vat, the glass paste is deposited at 1200 C over the tin. After a mechanical process of extension, sheets of different thickness are obtained at the opposite end of the vat, at 600 C. The plastic deformations induced in the glass are produced by towing the glass sheet along the vat. Because of the glass displacement, tin currents are generated, provoking several mass transportation phenomena, and therefore, different patterns of thermal energy are induced. These thermal conditions lead to heat imbalances that must be modelled with the aim of controlling the different mechanical and optical glass features to obtain the optimal quality. Up to now, these processes were controlled by human expertise, but for improving new prototype designs, a computer based methodology ought to be implemented.

The manufacturing modelling can be described in terms of two basic processes. First, a hydrodynamic process is necessary to predict the tin currents generated below the glass sheet. On the other hand, a thermal process must be modelled for forecasting thermal maps of both the glass sheet and the underlying tin layer. Both processes are interrelated with each other.

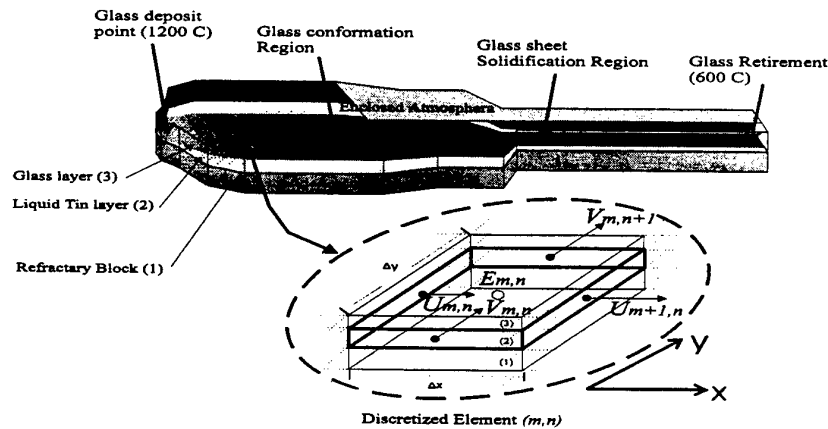


Fig. 1. Scenario representation: the production vat and its phases, the four-layer model, and a generic finite-difference discretized element (m,n) .

In order to achieve accurate results for the analysis and evaluation of new designs, some requirements are compulsory. The required degree of observability obliges us to obtain a detail level around 6 cm to reproduce all heterogeneous phenomena. This

implies obtaining a mathematical model based on a finite-difference scheme, obtaining a discretization mesh of the vat around 1000x150 elements. As figure 1 shows, this discretization produces a rectangular square-cell mesh, delimiting the computation domain. Obviously, the discretization phase takes into account all the elements involved in the physical process, such as cooling and heating elements, flux forming needs, tracking wheels to steer the glass sheet, etc. With this accuracy level, a non-optimised sequential program with the proposed modelling takes more than a week running in a HP9000/735 workstation, to simulate 20 minutes of a real process - time to obtain the steady state of the process. This amount of computation time makes the tool impractical and it can only be afforded by means of parallel processing.

1.2. The Mathematical Scheme

Due to the singularity of the process, appropriate numerical methods must be chosen. The unknowns of interest are the vectorial components of the tin velocity (U and V - in a discretized notation in figure 1), as well as the evolution of the tin level with respect to the free surface (identified by E). The glass and tin temperature evolution are also computed. We can describe the tin effect as a transportation process that can be approximated as a long wave behaviour, and permits important reduction in the modelling that represents such movement. This assumption is based on considering the vat length greater than its depth, allowing us to assume that flux evolution is mainly horizontal, and therefore that a 2D model is sufficient and quite reliable. Considering the discretization mesh in figure 1, to fully determine the long wave equation, two initial expressions must be introduced: the momentum flux equation (1) and the mass conservation for incompressible fluids equation (2), based on fluid dynamics found in [1][2]. These equations are expressed in the following form:

$$\rho \frac{Du}{Dt} = -\Delta p + pF + \frac{\partial \tau_{m,n}}{\partial x_n} \quad (1)$$

$$\Delta u = 0 \quad (2)$$

where:

u = tin vectorial velocity,

p = pressure

F = volume forces (gravity)

τ_{mn} = shear stresses in the n -th surface and m -th direction.

The thermal model solves a vertical-layer interaction problem, consisting in determining the thermal relationship between a four-layer based model. The vat is composed of four vertical layers, the refractory cooling block in the floor, the tin layer, the glass sheet and the enclosed atmosphere. To compute the temporal thermal evolution, the UPWIND method is used [13], with the typical conduction-advection equations involved in its solution. Moreover, both, the hydrodynamic and the thermal

processes, interact by means of the density gradient terms of the tin and the glass, that change continuously, associated with each discretized element.

In the hydrodynamic process, the longitudinal and transversal tin velocities - $U(m,n,t)$ and $V(m,n,t)$ - and the free-surface tin level - $E(m,n,t)$, are computed. With these unknowns solved, the tin density is affected by the thermal evolution, which also affects its velocity. The temperature in each cell, $T(m,n,z,t)$, is the next unknown solved in the same time iteration. The simulation process reaches its end, when the transient state has finished.

The model is based on explicit equations, and some non-linear elements can intervene, the time step of the simulation process is determined to assure the model stability and the final convergence. This time step is chosen applying the Courant and Peclet criteria [1]. These criteria also permit us to perform the thermal solution not in all the hydrodynamic time steps, because the thermal evolution is slower. The Courant and Peclet solution, associated with the mesh dimension, determines when both processes interact.

The discretization process transforms the differential equations (1) and (2) in two subsets of partial differential equation systems. One subset is that obtained for the X-dimension, and the other, for the Y-dimension. For each time step, U , V and E are computed for the "t+1" instant.

$$\frac{1}{\Delta t}(U_{m-1,n}^{t+1} - U_{m-1,n}^t) + \frac{1}{\Delta x}(E_{m,n}^{t+1/2} - E_{m-1,n}^{t+1/2}) + T_1 + T_2 + \dots = 0 \quad (3)$$

$$\frac{2}{\Delta t}(E_{m,n}^{t+1/2} - E_{m,n}^t) + \frac{1}{2\Delta x}(U_{m,n}^{t+1} - U_{m-1,n}^{t+1} + U_{m,n}^t - U_{m-1,n}^t) + \frac{1}{\Delta y}(V_{m,n}^t - V_{m,n-1}^t) = 0 \quad (4)$$

Non-linear terms appear and they are modelled as a collection of perturbations T_n , that are introduced in the system in each point of the mesh, for each instant "t". These terms represent different kinds of frictions, the density variations and other second-order terms. Notice that solutions in "t+1" consider instant "t" and "t+1/2". After expanding the discretized equations, following a bidimensional decomposition where the X dimension is divided in M elements, and the Y dimension in N, the ADI method is employed to solve the hydrodynamic process [3][4]. This scheme leads to solving M tridiagonal systems of linear equations of size $N \times N$ in the ADI X-sweep phase, and N tridiagonal systems of $M \times M$ elements in the orthogonal ADI-sweep.

2. Parallelization Approach

2.1. Initial Considerations

The high time consuming feature presented by the application discards any other computer-based approach except a parallelization scheme. To obtain a parallel algorithm for the ADI method, we propose to divide M cells along k processors, to solve the tridiagonal systems in the X dimension, performing the communication between them, dividing, once again N cells into k processors, and repeating the Y -sweep phase, collecting the solutions in a new inter-processor communication, and initialising another time step.

Prior to the parallelization task, an optimized sequential code was developed with the aim of generating a good reference to make further comparisons, such as speed-up, time execution bounds, etc. Production Fortran 77 was the chosen programming language, and both parallel programming paradigms were designed for the application, i. e., a shared memory code, as well as a message-passing one.

To analyse the parallel strategies, we identify the computation and communication phases. First, a domain decomposition is performed to assign each processor a sub-domain. On the other hand, in the inter-processor communication, an all-to-all personalised communication pattern [5] arises as the bottleneck of the algorithm.

2.2. The Shared-Memory Scheme

The strategy of programming the parallel tasks in the code, i.e. ADI and UPWIND methods, was done via loop partitioning. This SPMD approach is quite easy to perform if no dependencies arise among sub-domain partitions. In our case, each dimensional sweep can be done totally free of domain border dependencies to compute the solution in " $t+1$ ". Processors only need already computed border data, i.e., from the previous iteration, and can find these data in global memory. Therefore, in each time step, if the domain has been decomposed into $M \times N$ elements, the following steps are performed:

1. Consider M cells in the X dimension, each processor computes M/k tridiagonal systems of size $N \times N$. $U(t+1)$ and $E(t+1/2)$ are solved.
2. After this first sweep, all processors must await the other processors in a barrier.
3. Now, processors perform the Y sweep with N/k tridiagonal systems of size $M \times M$. $V(t+1)$ and $E(t+1)$ are computed.
4. When this sweep is performed, another barrier point is reached.
5. If Peclet's condition is true, the thermal process must be performed:

- 5a. A X-dimension partition is done. Each processor takes N/k portions of the temperature matrix T to solve the four-layer model in the Z direction, computing $T(m,n,z,t+1)$.
6. While the steady-state is not reached, continue with point 1.

The shared-memory paradigm allows an explicit parallelization of the regions computed with the creation of dynamic threads on each processor of the platform, using high-level directives via DOACROSS. The following portion of code is representative to express the X-dimension partition.

```
C*$* ASSERT CONCURRENT CALL

C$DOACROSS LOCAL (I)

      DO 110 I=1,M

          CALL LINX(I,T,DDX,DDY,IT,EDDY,EDDYZ)

110      CONTINUE

C$PAR BARRIER
```

The compiler interprets these directives format as concurrent calls to the `sproc()` routines [14], spawning parallel threads. The function, `sprocs()`, is a better implementation of the `fork()` routine, allowing the creation of light-weight processes. However, the main drawback is the impossibility of porting this code across other shared-memory platforms. Future work will go in the direction of developing a portable shared-memory code using POSIX threads.

2.3. The Message-Passing Implementation

With the aim of assuring code portability and for comparing other features, such as programming efforts, communications costs, and several architecture evaluations, we designed the same application using the distributed-memory parallel programming paradigm, using message passing.

We used the MPI communication library [6]. The programming effort is clearly higher versus shared-memory. But a wide spectrum of architectures can exploit the message-passing version, from workstation or PC networks, through real distributed memory machines as well as shared-memory architectures (either SMPs or CC-NUMAs) [7].

The core algorithm is basically the same as that described in section 2.2, except in points 2 and 4. The need of performing an explicit data transfer, via messages, applying an all-to-all communication pattern, obliges stalling the computation phase to initiate the collective data exchange. Figure 2, depicts one matrix distribution -

although this is valid for four matrices U, V, E and T. In this figure, let us assume a four-processor machine performing the X-sweep (figure 2a). The shadow sub-matrix is assigned to processor 2. When it has finished its computation phase, before performing the Y-sweep, processor 2 must send the blocks that the other processor will need. In the Y-sweep phase, processor 2 is assigned with the shadow portion of the matrix (in figure 2b). Only one common sub-block is owned by it, and the others must be received, as the figure shows. The all-to-all process is repeated when the Y-phase has finished, to continue with another time step.

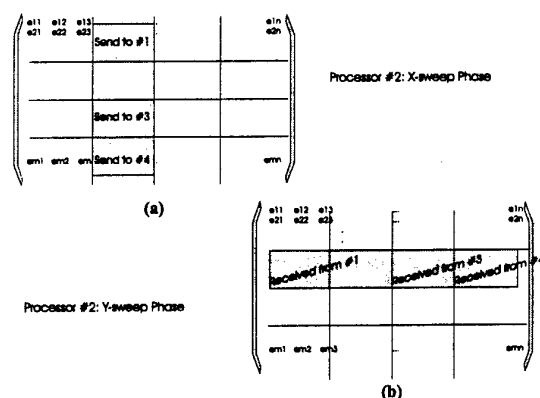


Fig. 2. Matrix decomposition. Computation domains between communication phases.

This inter-processor exchange is made by means of the usage of the all-to-all MPI primitive, and has been the major programming effort, since from the performance point of view, all this process penalises the execution time. Indeed, the basic portion of information to send is a sub-column (Fortran matrix storage) and before the beginning of a send message, as well as for the received message, suitable buffers must be allocated in the destination nodes. Then, each processor calls $M/(k-1)$ all-to-all primitives, where both receive and send communications are controlled by the MPI primitive.

2.4. Description of the Target Machines

A wide variety of machines have been tested under the parallel codes, with a double function. First, the code constitutes a high-communication demanding application, therefore, several remarks can arise from the behaviour of such architectures. Additionally, the lack of portability shown by the shared-memory version is covered by the message-passing code that has been ported without problems. But a poorer performance is achieved by the latter version.

In Table 1, there is a summary of the different machines, with their main features. The purpose of choosing these architectures is based on their different interconnection networks, and it permits the evaluation of the different performances of an ADI problem. A deeper discussion of these machines is presented in section 3.

	Processors	Interconnection Network (BW)	OS	Compiler
Power Challenge	16 R10000 (195 MHz)	POWERpath2 Bus (1.2 GB/s)	IRIX 6.2	MIPS Pro 7.0
Origin 2000	32 R10000 (195 MHz)	Hypercube	IRIX 6.4	MIPS Pro 7.2
Origin 200	4 R10000 (180 MHz)	2 linked-crossbar (1.2 GB/s)	IRIX 6.4	MIPS Pro 7.2
IBM SP2	32 POWER2 RS6000 (66 MHz)	Least Common Ancestor Network (80 MB/s)	AIX v.4	3.2.3
NOW Pentium	4 PentiumPro (200 MHz)	Myrinet, (1.28 Gbit/s)	Linux 2.0.32	egcs-pg77.1.0.2
Quad Pentium	4 PentiumPro (200 Mhz)	450GX Chipset Bus (512 MB/s)	Linux 2.1.79 SMP support	g77.0.5.22

Table 1. Main features of the Parallel Machines under study.

3. Results and Discussion

The first results we want to present are those in figure 3. The reason is to see how the application scales with the addition of more processors. The shared-memory code does not improve performance with more than 20 processors, for a problem of 1000x150. It is important to note that both the Power Challenge and the Origin 2000 exhibit the same execution time and the same speed-up. As they have very different networks, this behaviour allows us to conclude that the communication effects of the application still do not stress the two machines, and the loss in the performance is mainly due to a significant reduction in the computation-to-communication ratio per processor. Therefore, with more realistic use of the hardware resources, for this problem size, a 4-processor Origin 200 and 12-processor Power Challenge, show reasonable performance results. Comparing with the optimised sequential code, that takes more than 20 hours, we obtain 3.5 hours using 8 processors in the Power Challenge, obtaining a speed-up of 6.3.

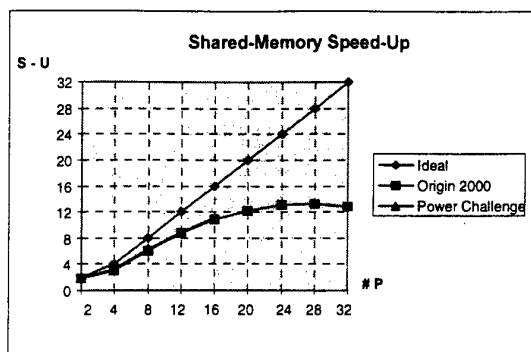


Fig. 3. Speed-up is presented for the shared-memory version, for a discretized grid of 1000x150. The machines are a 32-processor Origin 2000 and a Power Challenge with 16 processors.

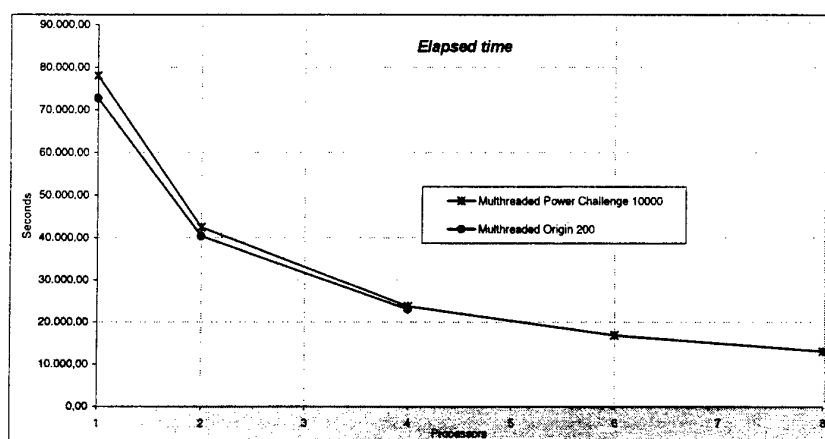


Fig. 4. Elapsed time is presented for the shared-memory version, for a discretized grid of 1000x150. A CC-NUMA 4-processor Origin 200 and a SMP Power Challenge with 12 processors are the target architectures.

The Power Challenge has 12 processors connected by a bus. This architecture permits a small degree of scalability, because the bus bandwidth is quickly saturated as the number of processors is increased. The Origin 200 is based on a quite different architecture. The machine groups two processors by nodes. In each node, there is a high-bandwidth crossbar that interconnects two processors with local memory, and the other nodes. So, the shared-memory application in a bus-based system is mainly limited by a problem of contention in its memory accesses. In CC-NUMA architectures, the application is limited by a data distribution problem. Therefore, as we have mentioned before, the main reason why the code shows the same

performance for these different architectures, is because neither contention problems nor high-latency data demands appear, for the working size of interest.

In a similar way, performance metrics for the distributed-memory version are presented in figure 5. This representation allows us to evaluate the effectiveness of the communication library implementation and its interaction with the architecture that supports it. Several conclusions arise from these performance behaviours. The results can be divided in three sub-sets.

The performance achieved with these machines shows that communication requirements are dominating the execution time. It seems not to benefit from adding more than four processors in all the cases.

The execution time for the SGI machines shows that the Power Challenge architecture improves the execution time over the Origin 200. However, the former quickly saturates the bus when more than four processors are running the application.

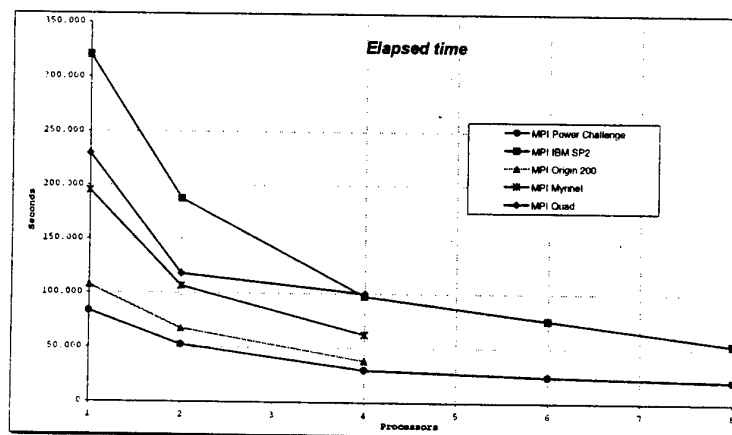


Fig. 5. Execution time achieved by the MPI version among the different architectures.

The second set of results are those presented by the PCs. On the one hand, a distributed-memory architecture composed of four Pentium Pro under Linux and a high-bandwidth low-latency Myrinet network [10][11]. The MPI library is implemented directly to a message-passing handler, (BIP [12]). Other implementations of the MPI have been tested previously over Myrinet, but have exhibited very poor results, because they do not provide specific support for the Myrinet hardware. Only the BIP library seems to be stable. Also, the compiler version for this platform generates code-optimisation for Pentium Pro. On the other hand, a Quad Pentium SMP architecture is the other configuration. The MPI implementation is the MPICH, and therefore, this socket-based implementation shows poor results. In

addition, the SMP kernel for this machine is still unstable, and does not accept the compiler version for code-optimisation, so aggressive features of that processor are turned off.

Finally, the IBM SP2 results show a poor performance, in execution time, over all the other platforms, although the application scales well enough and exhibits better communication costs.

In order to quantify the communication costs for this benchmark, in figure 6 the percentage of execution time spent in inter-processor data exchange, can be seen. The small percentage present in the IBM SP2 and in the Origin 200 is quite significant. The basic reasons can be found in their real physically distributed-memory architecture. Also, the effects of bus traffic saturation appear. When more than four processors are used in the case of the Power Challenge, up to 55% of the total execution time, for 8 processors, is spent in the communication phases. So, only the advanced features of the R10000 are permitting the best execution among the other platforms. Bus traffic-saturation effects also appear in the Quad using four processors. This 58% of communication costs is mainly due of the MPICH implementation, that relies on the operating system to perform the communication primitives.

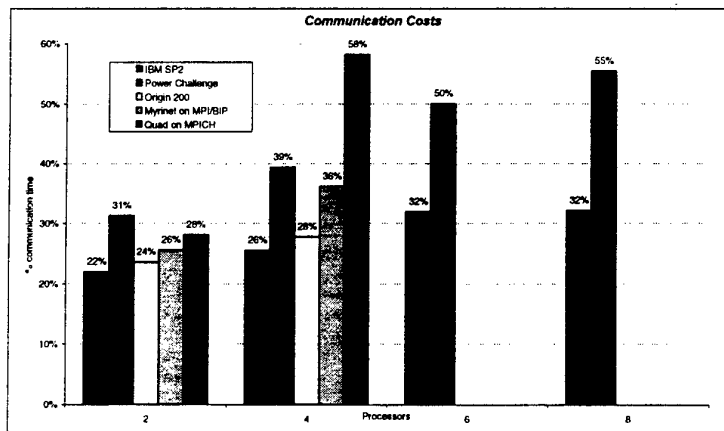


Fig. 6. Communication costs associated with the different MPI communication library implementations.

Finally, we present the speed-up behaviour as another important metric. Considering both versions of the application, a very good benchmark to analyse the interconnection network performance is by means of how the application scales. Moreover, the aim of testing all this wide spectrum of systems is to understand correct architectural design trade-offs.

In figure 7, good scalability, up to 6.3 over 8 processors is exhibited by the shared-memory version on the Power Challenge. But for this size, more than 16 processors do not improve the execution time (figure 3). The IBM SP2 presents the best speed-up, due to its low communication costs. However, the poorest execution time is achieved in this machine, because of the utilisation of a previous generation microprocessor.

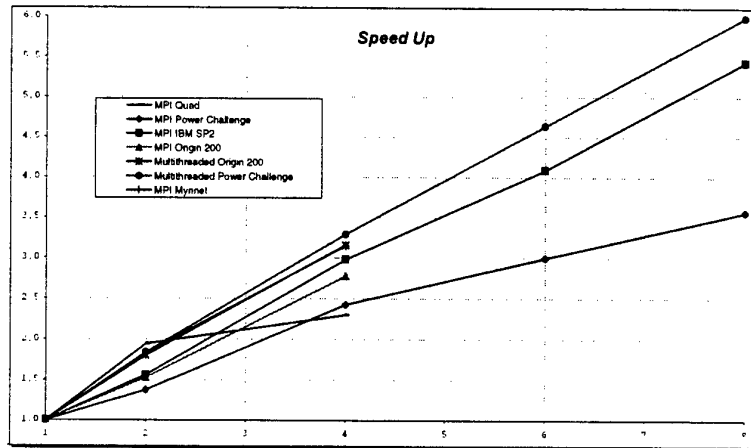


Fig. 7. Speed-up representation to evaluate application scalability bounds.

4. Conclusions and Future Work

To conclude, the industrial final-user is now exploiting the shared-memory program running on the Power Challenge, carrying out two simulations per day. New prototype manufacturing processes are achieved in such a manner that they take advantage of the best criteria for the designs that the simulations offer them. Higher product quality and better market competitiveness are the expected results by means this work.

In other investigative line, our research group is working in developing a shared-memory portable code. The effort is focused in getting multithreading via POSIX thread implementation with shared-memory support. Code portability, with the objective of wide spectrum machine usage, justifies this programming effort as we have seen when developing the distributed- memory code.

The singularity of the application means that the possibilities of using parallel processing are still unlimited. Parallel code development nowadays constitutes a multidisciplinary area where modellers, computer architects and software designers improve the best trade-off decision to achieve better performance, in a field where the convergence between hardware and software is still not sufficiently enlightened.

References

- [1] M.B. Abbott, Elements of the Theory of the Free Surface Flows., De. Pitman 1979.
- [2] C.G. Koutitas, Mathematical Models in Coastal Engineering, Pentech Press, 1988.
- [3] R.S. Varga, Matrix Iterative Analysis, Prentice-Hall, 1962.
- [4] G.H. Golub, C.F. Van Loan, Matrix Computations, 3rd Edition, The Johns Hopkins University Press, 1996.
- [5] S. H. Bokhari, Multiphase Complete Exchange on Paragon, SP2 and CS-2. IEEE Parallel & Distributed Technology, Fall 1996.
- [6] MPI Forum. MPI: A Message-Passing Interface Standard. June 1995.
- [7] J.L. Hennessy, D.A. Patterson, Computer Architecture, A Quantitative Approach, 2nd Edition, Morgan Kaufmann, 1996.
- [8] D. E. Lenoski, W. Weber, Scalable shared-Memory Multiprocessing. Edit. Morgan Kaufmann, 1995.
- [9] J. Miguel, A. Arruabarrena, R. Beivide, J. A. Gregorio, Assessing the Performance of the New IBM SP2 Communication Subsystem. IEEE Parallel & Distributed Technology, Winter 1996.
- [10] D.E. Culler, L.T. Liu, et al., Assessing Fast network Interfaces. IEEE Micro, February 1996.
- [11] C.L. Seitz, Myrinet - A Gigabit-per-second Local-Area-Network. IEEE Micro, February 1995.
- [12] Software, documentation and support available in <http://lhpc.univ-lyon1.fr>
- [13] R. Peyret, T.D. Taylor. Computational methods for fluid flows. Ed. Springer-Verlag. Berlin. 1982.
- [14] Irix 6.2 sproc man page. Silicon Graphics.
- [15] I. Scherson, C. K. Chien. Least Common Ancestor Networks. VLSI Design, vol. 2, no. 4, 1995.

Neural Classifiers Implemented in a Transputer Based Parallel Machine

J. M. Seixas¹, A. R. Anjos¹, C. B. Prado¹, L. P. Calôba¹, A. C. H. Dantas¹
and J. C. R. Aguiar²

¹ COPPE/EE/UFRJ, C.P. 68504, Rio de Janeiro 21945-970, Brazil

² CEPEL, Rio de Janeiro, Brazil

Abstract. A transputer based parallel machine is used as a development platform for fast neural signal processing applications in physics and electricity. The 16 node machine houses 32-bit floating point digital signal processors running as coprocessor for the transputers, so that signal processing can be optimized. The application in physics consists in a prototype of an online validation system for a high event rate collider experiment, which is implemented using neural networks for physics process identification. In electricity, a nonintrusive load monitoring system for household appliances is developed using a neural discriminator to identify seven groups of appliances.

keywords: parallel processing, neural networks, classifiers, principal component analysis, real-time systems.

1 Introduction

Neural networks find a vast area of applications in signal processing domain [1]. In particular, as classifiers, neural networks have been extensively used due to their ability in combining high classification efficiency and processing speed [2]. As inner products are the main mathematical operations required by the neural processing during the production phase, neural classifiers can be efficiently implemented in commercial programmable devices, such as digital signal processors (DSPs). Ultimate limits in processing speed can be reached if the natural parallelism of neural networks is explored [3]. Therefore, when both performance and speed are of concern in a project, neural networks may be considered an efficient solution.

In this paper we describe two applications of neural processing in a parallel computing environment. In the first one, an online validation system is designed for a high-event rate collider experiment in high-energy physics, which is being developed at CERN (Switzerland). In this experiment (LHC), bunches of particles will collide in periods of 25 nanoseconds, so that a large amount of experimental data will be produced [4]. However, events with physics significance will be extremely rare. Thus, the incoming data flow needs to be reduced by a highly sophisticated online validation system for deciding whether a given event should be discarded or stored by the data acquisition system. The second application involves the design of a nonintrusive electrical load monitoring system

for household appliances. As household appliances respond for a significant fraction ($\sim 25\%$ in Brazil) of the total demand in power consumption, the knowledge of the consumption profile of this segment is valuable for energy conservation and alleviation of the electrical system in peaking periods.

Both applications described in this paper are developed for the TN-310 system, a multiple instructions multiple data parallel computer with a distributed memory architecture [5]. The system (see Figure 1) houses 16 nodes based on T9000 transputers, which communicate with each other by means of a network of C104 chips [6]. Each node has access to the communication network through four high speed (100 Mbits/s) serial links (DS-links). For optimizing signal processing applications, the system includes fast 32-bit floating point DSPs (ADSP-21020) running as coprocessors for the transputers [7]. For this DSP, every instruction is executed in a single cycle (40 nanoseconds). In terms of memory, each node comprises 256 kbytes SRAM used as shared memory, to transfer data to and from the DSP, and 8 Mbytes of T9000 private DRAM. The DSP can be programmed from the T9000 through C runtime library calls.

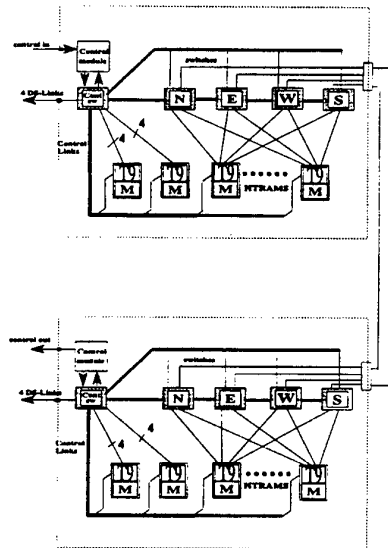


Fig. 1. The basic architecture of the TN-310 system.

The T9000 is a 32-bit microprocessor that exhibits multiprocessing capabilities. Communication between processes on different processors takes place over virtual channels. From the point of view of the applications, these communication facilities of the T9000 look attractive. Moreover, as the TN-310 includes a

fast DSP on each node, the required speed for running the neural processing can be achieved in a such transputer based machine.

The application development makes use of the C toolset environment, in order to achieve ultimate speed. This software layer features hardware configuration and a set of libraries and development tools to support ANSI C programming.

Developing an application in this environment comprises two phases. In the first, the user configures the available hardware according to the application. This phase describes the parallel application in terms of number of processors, amount of memory for each processor, interconnection matrix of processors, the use of the cache memory and control signals. In the second phase, the user develops C language codes according to the resources available from the hardware configuration of the first phase.

In our case, the TN-310 system is accessed through a PC running MS-DOS and MS-Windows.

2 The Applications

For both applications to be described, fully-connected feedforward neural classifiers were designed. These classifiers were trained on preprocessed data by means of backpropagation method [2]. The preprocessing phase was introduced in order to reduce the dimensionability of data input space, so that more compact classifiers could be developed.

2.1 Online Validation System

The LHC (Large Hadron Collider) will produce event rates up to 100 MHz but only very rare events will have physics significance to the experiment. In order to remove such deep background noise that hides potentially interesting events, an online event validation system is being designed as a multi-level triggering system [8]. In the first level, a very fast algorithm will be capable to reduce the event rate to 100 kHz. The second-level triggering (LVL2) system will only act on events that passed the conditions of the previous level. Not all regions of the detectors contain valuable information for a given event, so that only restricted areas in the detector (known as Regions of Interest - ROI) will be moved by the first-level system to the LVL2 system. This will alleviate bandwidth requirements on the LVL2 system, which is expected to achieve a further reduction factor of 100 in the event rate. The surviving events will be analyzed by a third level trigger and only 10 or 100 events per second will be moved to permanent storage.

The prototype being implemented in the TN-310 system concerns the LVL2 operation. The architecture used splits the LVL2 operation into two phases [9]. In the first, raw detector data is translated into features capable to efficiently identify the relevant physics processes. This feature extraction works on ROI information provided by the detectors that participate in the LVL2 decision: calorimeters (for energy measurement), trackers (for image display of inner interactions) and muon chambers (for muon detection). Next, the global decision

phase correlates detector information for analysis refinement. Features are combined to compute the probability of a particle to be found in a given ROI, so that physics processes can be identified.

Both phases may be performed by neural processing [10, 11]. For the calorimeter subsystem, feature extraction was performed by grouping cells of deposited energy in a ROI, and feeding such grouped cells into the input nodes of a neural network that performs electron (signal)/jets (background noise) separation.

Figure 2 shows how cells are grouped together. Thicker lines define the border of each region, and cells belonging to a region have their energies added up to form group sums. As outermost regions play an important role in the discrimination process but have their energy values masked by the substantially higher energy level of the center region, optimum weighting factors for these regions were determined by integrating the search of such optimum weighting profile to the training phase of the network [12].

This grouping scheme is capable to combine efficiently performance and compactness, as it achieves high discrimination levels and also reduces substantially the number of input nodes of the network (now ten, instead of the original 121 components of the ROI). For the implementation of the neural feature extractor, the neural network comprised 3 hidden nodes and a single output neuron, so that electron/jet (of particles) discrimination could be efficiently performed (only 7.3% of jets were misclassified as electrons for a 97% electron efficiency). For the other three detectors involved in the LVL2 decision scheme, a simulation of current classical algorithms was implemented [8].

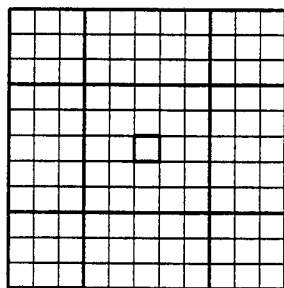


Fig. 2. The grouped sum structure.

For the global decision phase, it was explored the capability of neural networks in correlating information in a multidimensional feature space. A set of twelve features from the detectors that participate in the LVL2 decision was fed into the neural classifier, which comprised six neurons in the hidden layer and four output nodes, so that electrons, pions, jets and muons could be detected (see Figure 3). The normalization of the feature vector was performed by means

of fixed factors, which were obtained by computing the mean value of the data distribution (restricted to the training set) for each feature and allowing an additional 2σ tail. Maximum probability defined the winner class for a given pattern fed into the input nodes of the classifier.

Simulated data sets for the second-level trigger operation at LHC conditions were used for training the networks [13, 14].

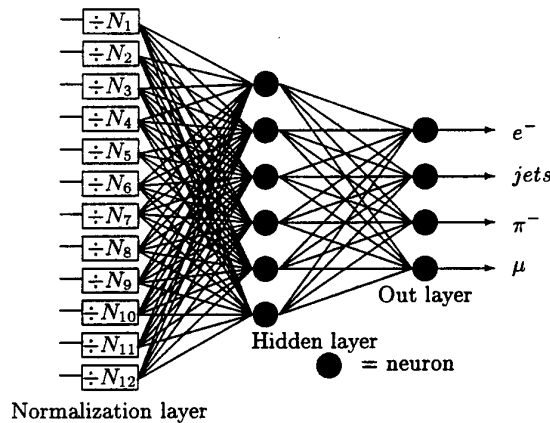


Fig. 3. The global decision network.

2.2 Electrical Load Monitoring System

Both transient and steady-state information were used to characterize data acquired from seven groups of household appliances: refrigerating, resistive heating, universal motor, ventilating, consume electronics, incandescent lamp and fluorescent lamp.

Starting from the time each appliance had been switched on, current signals were sampled (500 samples per second) from the AC line by means of a digital storage oscilloscope for a 2 second acquisition window. Steady-state response provided current and phase angle information and measurements were made after a minimum of 2 seconds operation [15].

The envelope of the transient signal is defined by the 60 Hz fundamental frequency of the AC line and features from its shape can be extracted by retaining the current peaks. From the original 1024 samples, 200 peaks were retained. A principal discriminating analysis was performed on such samples. This analysis aimed at finding the most discriminating components on data input space, so that classification can be achieved with minimal number of hidden nodes in the network [16].

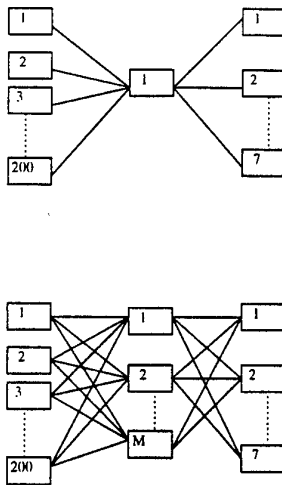


Fig. 4. Basic topology for the principal discriminating analysis.

Figure 4 shows the network topology that extracts the principal discriminating components and simultaneously identifies the classes of household appliances. Starting from a single neuron in the hidden layer (arrangement shown at the top of Figure 4), the network is trained to obtain maximum discrimination efficiency. Next, a second neuron is added to the hidden layer of the network. The resulting network is trained in such a way that the weighting vector that connects the input nodes to the first neuron in the hidden layer is kept fixed during the training procedure, as it represents the first component already extracted in the previous phase. All the other synaptic weights in the network are changed according to the backpropagation method, including the weight vector that connects the first neuron of the hidden layer to the output layer. This is to allow the network to combine in the best way the discriminating components, as a new component becomes available in this phase. The training procedure continues in this way, by adding a new neuron in the hidden layer, freezing the components previously extracted and allowing the rest of the synaptic weights to be trained until the addition of new components does not result in an improvement on the overall discrimination efficiency.

For this application, only two neurons in the hidden-layer were needed. Each output node was assigned to a group of equipments and maximum probability was used to detect the winner class for a given input.

3 Implementations

The system implementation for both applications made use of a master node to perform communication with the outside world and to supervise the continuous

distribution of data through slave processors. The slave processors act as feature extractors or global decision units (gdus) for the physics application (Figure 5), and perform preprocessing and neural classification for the load monitoring system. For the validation system, a local network (split into two processing nodes) is used to label and group data from the feature extraction phase and to transmit features to the global decision layer.

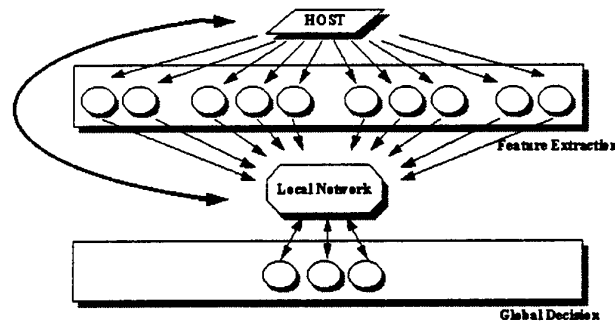


Fig. 5. Simplified scheme for the implementation of the validation system.

The data parallelism approach that is being used intends to minimize the communication overheads of the TN-310 system. As previously mentioned, the system uses a fast switching network (based on C104) for fast packet distribution among processors. When an information is to be transmitted, the emitter point sends a message to its local virtual channel processor (VCP) on the T9000, indicating the receiving address and the data size to be transmitted. The VCP splits the data into a number of packets of 32 bytes and each packet carries a header that indicates the receiving address and the routing needed. Then, the packet is sent to the first switch in the predefined routing which interprets (and then removes) the first subheader and sends the packet to the next node in the routing. At the final destination, each packet is acknowledged to allow the transmission of a new packet. Figure 6 illustrates this scheme.

The minimum time required for packet (32 bytes) transmission was measured to be $\sim 7 \mu s$. Therefore, as neural networks for both applications are relatively compact (and fast: feature extraction for the calorimeter is achieved in $10 \mu s$), data communication time can be considered quite significant to the overall processing speed. Consequently, data parallelism will minimize dependencies among nodes and the speedup of the applications will be maximized.

As communication time represents the bottleneck of applications in this environment, it may be useless to develop the application using all resources of the machine. For instance, when data are distributed to, say slave # M in the parallel processing chain, the first slave may have finished its processing and become free. Thus, further nodes added to the chain will not improve processing speed.

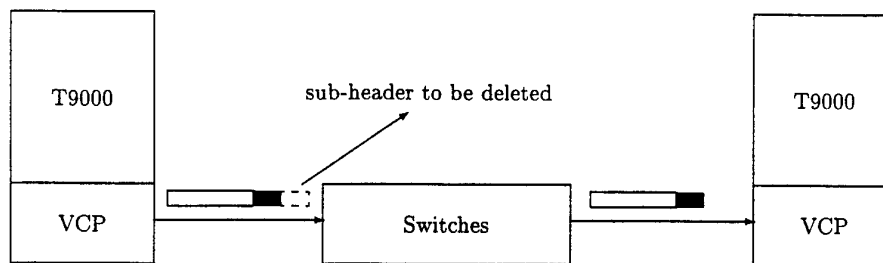


Fig. 6. Routing packets in the TN-310 system.

This fact was explored to develop application using minimal machine resources. On the other hand, as slaves become free, a two distributor scheme may be tried to compensate for communication overheads. Using twin data parallelism structures, an improvement in processing speed and parallel efficiency may be achieved.

The activation function (hyperbolic tangent) of the neural networks was implemented by means of a look up table, in order to achieve shorter computation times. The fixed step table makes possible a correspondence between the address of a point in the table and its abscissa. Thus, the look up table is fast and constant in execution. Moreover, table size is reduced as abscissa values are memory addressed and only ordinates are stored.

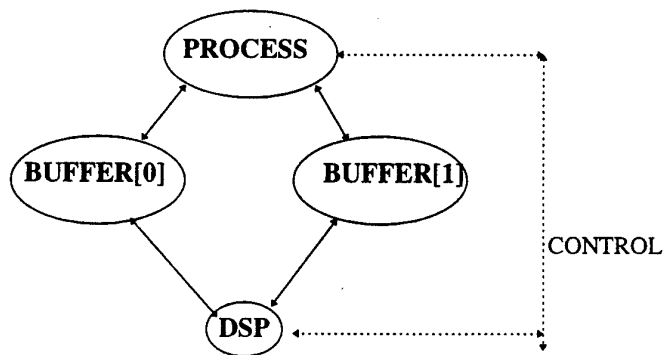


Fig. 7. The way the DSP can be accessed from the T9000.

The sampling resolution for building such table for the applications and which provided the full reproduction of the simulated network operation with minimum memory requirements was achieved to be 0.01. Saturation of the activation func-

tion was considered to be reached at 7 for both positive and negative arguments. This approximation of the behavior of the gain function also did not deteriorate the performance of the network and reduced considerably memory length requirements.

For a multiple processor application, a user process placed on one processor may access the DSP on its site and not any other. The DSP library available by the TN-310 system is written in C language and provides signal processing and mathematics functions. The way the user process accesses its accompanying DSP is shown in Figure 7. The process has access to the two banks of shared memory and controls the access protocol to the DSP. An alternate scheme is an efficient way to use the buffers: the user process loads some commands, or reads the previous results, in one buffer, while the DSP computes de previous commands of the other buffer. This is supported by a *DSPbufSwap* procedure that returns a buffer number. When the procedure returns a buffer number, the user process can access this specific buffer to read its contents or to write commands into it. Then, the process releases the buffer by using the *DSPbufSwap* and the DSP starts to compute the given command. The DSPs were used to compute inner products required by the neural processing and required preprocessing.

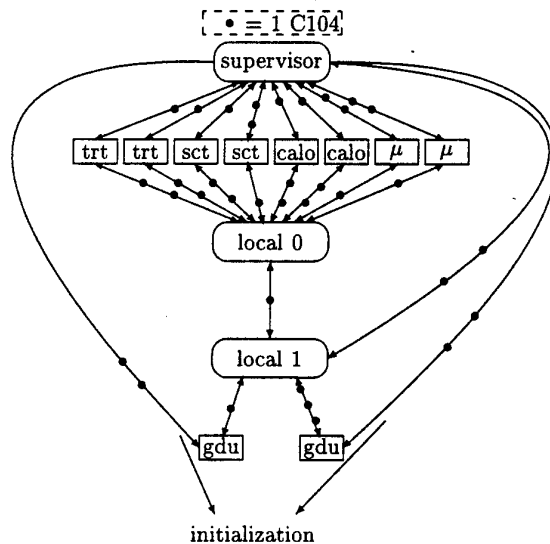


Fig. 8. The validation system implementation using minimal machine resources.

The prototype of the validation system for physics was implemented using 13 nodes (see Figure 8) and was capable to cope with a 2.6 kHz input frequency (speedup factor of 4.4). The nodes labeled *trt* and *sct* refer to subdetectors of the

tracking system, *calo* to the calorimetry system, and μ to the muon chambers. For each processing layer only two nodes were needed, as data communication between the supervisor (master) node and slaves required significant time when neural processing speed is considered (as mentioned above). For instance, data size for the calorimeter required more than 100 μ s in terms of transmission time, but the neural feature extraction for this detector is performed in 10 μ s.

The resulting system can be considered to emulate a vertical slice of the actual second-level triggering system, which will be running in practice close to this speed. Particle identification above 94% was achieved by the system.

For the load monitoring system, principal component analysis and classification was capable to run in less than 100 μ s (speedup factor of 8.6). Here, as data input vectors comprised 200 components, data transmission required barely as much time as data processing. Therefore, only 6 processing nodes were needed to implement the system. Such consideration allowed the use of a two distributor configuration to this application, as mentioned above. Figure 9 shows this configuration, where a master node interfaces with the outside world and transmits data to its slaves and the other master node. Over 100 different pieces of equipment studied, the system was able to classify correctly more than 84% of the sample.

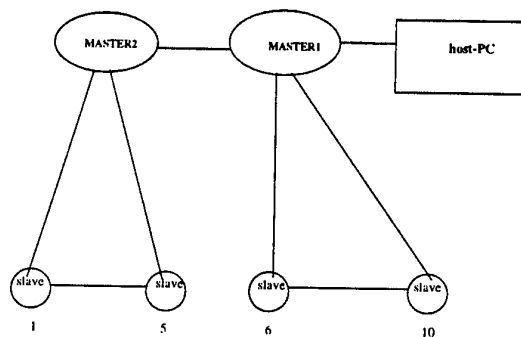


Fig. 9. Basic implementation scheme of the load monitoring system.

4 Conclusions

A transputer based parallel machine (TN-310 system) was used as a development platform for neural processing applications. The applications explored the main features of this machine, which combines the ability of the T9000 transputer in managing communicating processes running in parallel and the signal processing speed of the ADSP-21020 digital signal processor. Although significant communication overheads were observed, the technology used proved to be flexible enough to accommodate different sorts of applications.

The applications concerned a prototype of an online validation system for a high event rate collider experiment in particle physics and a nonintrusive electrical load monitoring system for household appliances. The design procedure for both applications involved preprocessing methods (a topological mapping or principal component analysis) and the search for the optimization of machine resources according to application requirements. Neural processing was implemented by using a look up table for the activation function and runtime library calls to the DSPs.

In order to compare technologies, the applications are being moved to a faster DSP (ADSP-21060, 25 ns instruction cycle time) that has integrated multiprocessing features. This DSP based solution goes in the direction of using more cost effective commercial computing platforms, as PC-compatible commercial boards are readily available in this technology.

References

1. Masters, T.: *Signal and Image Processing with Neural Networks*. John Wiley & Sons (1994).
2. Haykin, S.: *Neural Networks. A Comprehensive Foundation*. Macmillan (1994).
3. Nelson, M.E., Furmanski, W., Bower, J.M.: *Simulating Neurons and Networks on Parallel Computers*. In *Methods in Neuronal Modeling*, edited by Koch, C., Segev, I. MIT Press (1992) 397-437.
4. <http://wwwlh01.cern.ch/>
5. Telmat Multinode: TN-310 System Manual and Training Set. France (1995).
6. Inmos Limited: *The T9000 Transputer Hardware Reference Manual* (1993).
7. Analog Devices, Inc.: *ADSP-21020/21010 User's Manual* (1993).
8. The Atlas Collaboration: *Technical Proposal for a General-Purpose pp Experiment at the Large Hadron Collider at CERN*. CERN/LHCC/94-43 (1994).
9. Hansen, J.R.: *Local-Global Demonstrator Programme for the ATLAS Second Level Trigger*. International IEEE Real Time Conference, Beaune, France (1997).
10. Seixas, J.M., Calôba, L.P., Souza, M.N., Braga, A.L., Rodrigues, A.P.: *Neural Second-Level Trigger System Based on Calorimetry*. *Computer Physics Communications*, Volume 95 (1996) 143-157.
11. Seixas, J.M., Santos, R.W., Calôba, L.P.: *Implementing a Neural Second-Level Trigger System on Digital Signal Processor Technology: The Global Decision Problem*. First Int. Workshop on Electronics for LHC Experiments, Lisbon, Portugal (1995) 324-327.
12. Seixas, J.M., Calôba, L.P., Linhares, R., Kastrup, B.: *Neural Feature Extraction for Calorimeters Based on Optimum Weighting Procedures*. *Nuclear Instruments and Methods*, A389 (1997) 146-147.
13. Klyuchnikov, G. et al.: *A Second-Level Trigger Based on Calorimetry Only*. CERN/ATLAS/DAQ-007 (1992).
14. Bock, R. et al.: *Test Data for the Global Second-Level Trigger*. CERN/EAST 93-01 (1993).
15. Aguiar, J.C.R.: *Nonintrusive Identification of Residential Electrical Loads Using Neural Networks*. M.Sc. Thesis, Federal University of Rio de Janeiro, Brazil (1996).
16. Calôba, L.P., Seixas, J.M., Pereira, F.S.: *Neural Discriminating Analysis for a Second-Level Trigger System*. International Conference on Computing in High Energy and Nuclear Physics, Rio de Janeiro, Brazil (1995) 870-874.

Algorithm-Dependant Method to Determine the Optimal Number of Computers in Parallel Virtual Machines

Jorge Barbosa^{*1} and Armando Padilha¹

¹FEUP-INEB, Praça Coronel Pacheco, 1, 4050 Porto (P)
e-mail: jbarbosa@tom.fe.up.pt

Abstract. Presently computer networks are becoming common in every place, connecting from only a few to hundreds of personal computers or workstations. The idea of getting the most out of the computing power installed is not new and several studies showed that for long periods of the day most of the computers are in the idle state. The work herein refers to a study aiming to find, for a given algorithm, the number of processors that should be used in order to get the minimum processing time. To support the parallel execution the WPVM software was used, under a Windows NT network of personal computers.

1 Introduction

A parallel computer is composed by a set of processors connected by one network according to one of several possible topologies (mesh, hipercube, crossbar, central bus, etc [8]). If the processors are connected in order to maximize their communication performance and together operate exclusively for the solution of one problem, then it is called a Supercomputer. If the processors are of the general purpose type, each one being a workstation connected by a general purpose network (e.g. Ethernet), then when they operate together for the solution of a given problem, it is called a Parallel Virtual Computer.

There are significant differences between a Supercomputer and a Parallel Virtual Computer, such as the interconnection network. The general purpose network allows only the communication between two processors simultaneously, and it could be also shared by other computers not belonging to the Parallel Virtual Computer, resulting in low communication rates. The fine grain parallelization, common in Supercomputers, becomes impractical in Parallel Virtual Computers, where medium or coarse grain parallelizations are used, at the program or procedure level.

The aim in the utilization of a Parallel Virtual Computer, as for a Supercomputer, is to reduce the processing time of a given program. This is achieved by utilizing execution cycles of several computers that would not be used in another way. The more conclusive measure of the parallelization performance

^{*} PhD grant BD/2850/94 from PRAXIS XXI programme

is the reduction of the processing time obtained, or equivalently the Speedup obtained.

This report presents a method to calculate the number of processors that should participate in the execution of a given algorithm according to its characteristics.

2 Interconnection Networks

A parallel computer is built of processing elements and memories, called nodes, and an interconnection network, composed by switches, to route messages between those nodes. Interconnection network topology is the pattern by which the switches are connected to each other and to nodes. The network topology can be classified as direct or indirect. Direct topologies connect each switch directly to a node, resulting a static network that will not change during program execution. Examples of static networks are the mesh, hypercube and ring.

Indirect topologies connect at least some of the switches to other switches, resulting dynamic networks that can be configured in order to match the communication demand in user program. Examples of dynamic networks are the multistage interconnection network, the fat-tree, crossbar switches and buses [7].

The interconnection network of a parallel virtual computer is similar to a bus as shown in Figure 1.

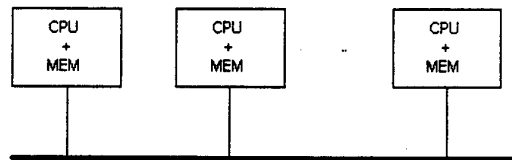


Fig. 1. Interconnection Network (Ethernet) of a Virtual Parallel Computer

The logical topology of an Ethernet provides a single channel, or bus, that carries Ethernet signals to all stations allowing broadcast communication. The physical topology may include bus cables or a star cable layout; however, no matter how computers are connected together, there is only one signal channel delivering packets over those cables to all stations on a given Ethernet LAN [10].

Each message is divided in packets of 46 to 1500 bytes of data, to be sent sequentially and individually onto the shared channel. For each packet the computer has to gain access to the channel. This division of a message into packets leads to a latency time for each message that is proportional to the number of packets into which it is split,

$$T_c = kT_L + \frac{nbytes}{w} \quad (1)$$

T_c being the total communication time for the given message, k the number of packets into which the message is split, T_L the latency time for one packet, $nbytes$ the message length in bytes and w the network bandwidth. For a particular system, equation 1 is used to estimate the value of T_L . For the network used in the Virtual Computers, M1 and M2, referred to in the results section, the mean value for T_L was estimated as being 500 microseconds and the packet size to be 1Kbyte.

3 Performance Measures

A Parallel Computer can be evaluated according to several characteristics, such as the processing capacity (Mflop/s), the network bandwidth (Mbytes/s), the processing capacity of each processor individually, the memory access method and time, etc. However, its performance is always referred to a given algorithm.

The ratio between the serial processing time and the parallel processing time is referred to as Speedup and reflects the gain obtained with the parallelization:

$$Speedup = \frac{T_{Serial}}{T_{Parallel}} \quad (2)$$

3.1 Speedup Limits

For a given problem, of a given dimension, there is a finite quantity of work required to be done in order to obtain its solution. Therefore, there will be a maximum number of processors to be used, above which there will not be any work to schedule for additional processors. Thus, the number of processors to be used and the maximum Speedup achieved for a given problem is limited by the quantity of work to be done. As an example, consider the addition of two vectors of dimension n , which involves n additions. If one uses more than n processors, the remaining processors will not have any work to do, resulting a maximum relative Speedup of n .

Amdahl [5] has defined a rule to demonstrate that the Speedup value is limited by the inherently sequential part of the program: let s represent the sequential part of the program, non parallelizable, and p the part of the program susceptible of being parallelized, that can execute with Speedup P in a computer with P processors, then the observed Speedup will be:

$$Speedup = \frac{s + p}{s + \frac{p}{P}} \quad (3)$$

$s + p$ being the processing time of the sequential program, and P the number of processors used. As an example, if the serial program runs in 93s in which 90s is susceptible of being parallelized, then $p = 90s/93s = 0.9677$ or 96.77% and s ,

the inherently sequential part, assumes the value of 0.0323 which is 3.23% of the code [6]. The sequential part is composed mainly by input/output operations.

From the speedup definition one can obtain its limit: as p approaches infinite, Speedup equals $1/s$. For the example presented above the Maximum Speedup is $1/0.032 = 31.25$ whatever the number of processors used, being useless to use more than 31 processors.

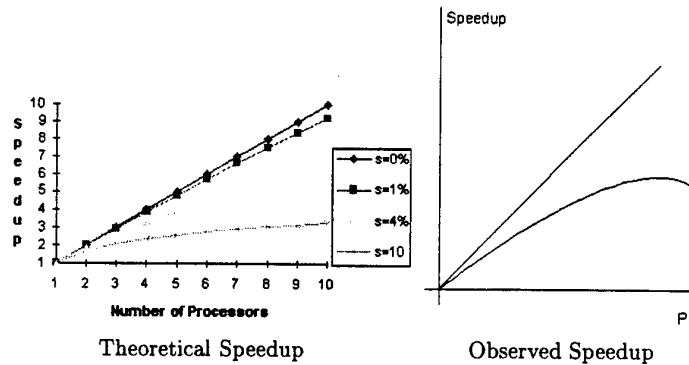


Fig. 2. Comparison between Theoretical and Observed Speedup

Figure 2 (left) shows the Theoretical Speedup for several values of s , which is assumed constant for a given algorithm, whichever the value assumed by P . The Amdahl law introduced two important factors in parallelism. First, it allows to have a more realistic expectation of the results that can be achieved, and second, it shows that for the Speedup to achieve high values, one must reduce or eliminate the sequential parts of a given algorithm [6].

Amdahl made s constant; however, in most algorithms, the increase in the number of processors leads to an increase in the communication overheads. If those are considered as bottlenecks and added to s , the Speedup behavior will be like the one shown in Figure 2 (right). The observed speedup presents a shape that increases until a given value of P is reached, after which it decreases. In conclusion, the ideal number of processors to be used for the solution of a given algorithm will be below of the number obtained by the limit of Amdahl's law.

3.2 The ideal number of processors

From the Speedup expression given above, its value will increase as the execution time of the parallel program decreases. Assuming that the parallel time is given by $T_p(n, P) = s(n, P) + p(n, P)$, as shown in Figure 3, for a generic algorithm, the processing time is composed by an initial operation for data distribution, followed by the time for parallel processing, including messages, and ending with an operation for collection of results by the master process. Then to get

the highest speedup, T_p has to be as low as possible, and the optimal value for P that satisfies this condition is such that, for a given algorithm, the increase of the serial component due to the addition of one more processor will balance the gain obtained in the processing time of the parallel component.

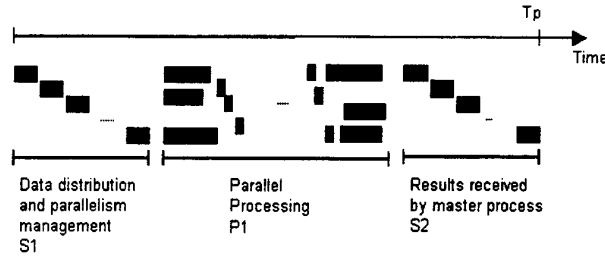


Fig. 3. Timing diagram of a parallel algorithm

The function representing the sequential component of the algorithm, $s(n, P)$, depends on the problem dimension, n , and on the number of processors used, P . It represents the input/output operations, the sequential code required to manage the parallelism and the communication overheads. Assuming an homogeneous computer network, where each processor has an individual processing capacity of $SM\text{ flop/s}$, a network bandwidth of $WM\text{ bytes/s}$, and n input data elements to be distributed, one gets the following expression:

$$s(n, P) = \frac{C_1 P}{S} + \frac{n}{P} \left(\frac{1}{W} + T_E \right) P + k_1 T_L + \left(\frac{1}{W} + T_E \right) b(P-1) + k_2 T_L (P-1) + \frac{k_3}{P} \left(\frac{1}{W} + T_E \right) P + k_4 T_L \quad (4)$$

In the expression above it was assumed that each process communicates only with the neighbor processes which is certainly not true for all algorithms. In practice the communication components have to be modeled for every algorithm. The first factor represents the time spent for the parallelism management, where C_1 is a constant dependant on the number of instructions being used. The second factor represents the time required to distribute the n elements of the input data among P processors, where T_E is the packing time per byte. The third factor represents the latency time of P initial messages required to distribute the input data, where k_1 is the number of packets required to send the data. The fourth factor represents the time spent by each processor in communications with the next processor for the parallel algorithm, transmission and packing for sending and receiving. The fifth factor represents the latency time of those messages per processor, k_2 being the number of packets required. The sixth factor represents the time required by the master process to receive the results from all processes,

k_3 being the number of bytes to receive. The last factor is the latency time for the results messages, where k_4 is the number of packets required.

The function representing the algorithm's parallel component, $p(n, P)$, depends also on the problem dimension and on the number of processors used. It can be calculated as:

$$p(n, P) = \frac{n\beta^\alpha}{PS} \quad (5)$$

β being the number of instructions computed α times for each element of n . From the addition of both expressions, $s + p$, one can obtain the total parallel processing time:

$$\begin{aligned} T_p(n, P) = & \frac{C_1 P}{S} + \frac{n}{P} \left(\frac{1}{W} + T_E \right) P + k_1 T_L + \left(\frac{1}{W} + T_E \right) b (P - 1) \\ & + k_2 T_L (P - 1) + \frac{k_3}{P} \left(\frac{1}{W} + T_E \right) P + k_4 T_L + \frac{n\beta^\alpha}{PS} \end{aligned} \quad (6)$$

For a given problem of dimension n and assuming the constants α and β are known, the minimum value for $T_p(n, P)$ is given by $\frac{\partial T_p}{\partial P} = 0$:

$$\frac{C_1}{S} + \left(\frac{1}{W} + T_E \right) b + k_2 T_L - \frac{Sn\beta^\alpha}{(PS)^2} = 0 \quad (7)$$

resulting for P the value

$$P = \sqrt{\frac{n\beta^\alpha/S}{C_1/S + k_2 T_L + (1/W + T_E) b}} \quad (8)$$

This expression shows that P is obtained by the square root of the ratio between the useful processing time by the time spent in communications and parallelism management. The value of P is then used to compute the expected processing time for the parallel program, T_P , which is then compared to the estimated serial processing time, T_S . If it happens that $T_P > T_S$ then the serial version of the algorithm is used.

3.3 Application to the Parallel Virtual Computer

The nodes of the Parallel Virtual Computer are composed by processors of different characteristics, mainly with respect to the processing capacity and memory available, forming an heterogeneous system. Therefore, the value of S , made constant in the computation of P , needs to be replaced by a value that represents the Heterogeneous Parallel Virtual Computer. Thus, S can be replaced by a weighted mean such as:

$$\bar{S} = \sum_{i=1}^M S_i w_i / \sum_{i=1}^M w_i \quad (9)$$

w_i being the weight given to processor i , defined as the ratio between the processing capacity of processor i and the capacity of the fastest processor in the Virtual Computer: $w_i = S_i/S_{MAX}$. With this weight the fastest processors have more influence, leading to lower values of P , and probably to the utilization of only the fastest processors. The distribution of computational load (l_i) is made proportional to the relative processing capacity of the i processor:

$$l_i = S_i / \sum_{k=1}^P S_k \quad (10)$$

4 Implementation

The parallel implementation of the image processing algorithms referred to in the results section is done under the WPVM software, which is an implementation of PVM for the MS Windows operating system, developed at University of Coimbra, Portugal [1, 2]. The software can be downloaded from <http://dsg.dei.uc.pt/wpvm>. WPVM offers the same set of functions as standard PVM [4] and allows the interaction between WPVM and PVM hosts.

5 Results

To validate the presented methodology, two image processing algorithms were implemented: a step edge detection algorithm [9] and an algorithm for histogram computation [3]. Also, two parallel virtual computers were used, M1 and M2, composed by the following processor capacities, in Mflops: M1={80, 80, 80, 80, 45, 45, 40, 40, 35, 35, 35} and M2={161, 161, 105, 91, 80}, with equivalent processing capacities, S1 and S2, of 68.8 and 129.7 Mflops, respectively.

For both algorithms the computational load is evenly distributed on the domain space, since the same operation is carried out for every input data element (an image pixel).

5.1 Step edge detection algorithm

Edge detection is an important subject in image processing because the edges correspond in general to objects that one wants to segment. The edge detector operator described in [9] is an optimal linear operator of a infinite window size. The operator is an infinite impulse response filter realized by a recursive algorithm. To find the filter response for each pixel, the components along x and y are first computed. Each of them requires four basic operations to be executed, as shown in Figure 4.

The operations are independent from each other; however, in each basic operation, the result for the previous pixel has to be known. This dependency is important in the context of data distribution, which should minimize the number of accesses to non local data. For this algorithm, a square blocked distribution

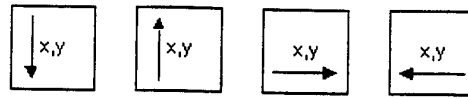


Fig. 4. Edge detection algorithm

(see Figure 5), requires more messages than the column or row blocked strategies. The data distribution selected was row blocked, because the image is stored row by row in the computer memory, requiring only one packing instruction to pack several contiguous rows. Row and column blocked distributions require the same amount of data to be transferred among processors.

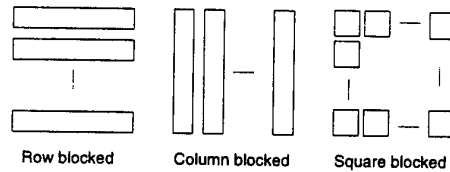


Fig. 5. Data distribution strategies

Due to the row blocked distribution, the result along columns has to be sent to the neighboring processors. The parallel implementation of the algorithm should allow each processor to start any of the four basic operations as soon as they have the data to start, avoiding the idle state. Figure 6 shows an optimized timing diagram for 3 processors, where processing starts as soon as the data is available and processors give priority to the operation results that others may be waiting for; in this case the priority is given to operations along columns.

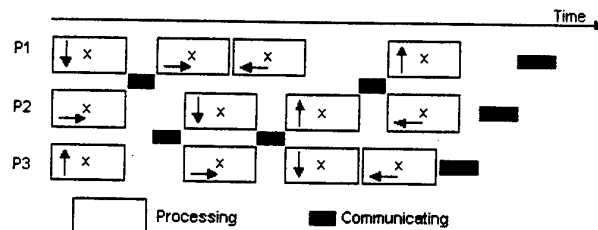


Fig. 6. Timing diagram for 3 processors

By measures made in tests of the parallel algorithms, the parameters that characterize them were obtained as shown in Table 1. The interconnection network, a Fast Ethernet at 100Mbit/s, has a transmission time of $0.08\mu\text{s}/\text{byte}$. The packing time T_E , of value $0.07\mu\text{s}$, was measured indirectly.

Algorithm	β^a (Mflop/kb)	k_3	b (bytes)	C_1 (Mflops)
Edge Det.	1.333	$\lceil \frac{b}{1024} \rceil$	$145 \times \text{ncolumns}$	0.8
Histogram	0.150	$\lceil \frac{b}{1024} \rceil$	$\frac{N}{P}(P-1)$	0.8

Table 1. Algorithm parameters

The edge detection algorithm was run in machine M1 with 68.8 Mflops, the master process being in a 80 Mflops computer. By replacing all the parameters in the expression of P , for an image of 64 kb (256×256), one gets:

$$P = \sqrt{\frac{64 \times 1.333/68.8}{0.8/80 + 0.5 \times 10^{-3} \lceil \frac{145 \times 256}{1024} \rceil + 145 \times 256(0.15 \times 10^{-6})}} = 6.03 \quad (11)$$

The serial component due to the parallelization management is run in the master process and therefore it is the master speed that divides the constant C_1 .

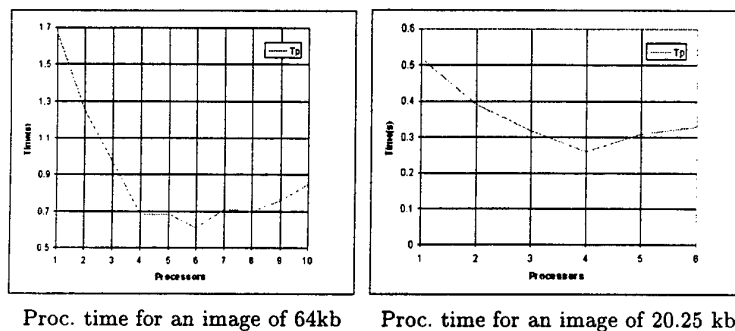


Fig. 7. Processing time of the parallel algorithm in a Parallel Virtual Computer

Figure 7 represents the processing time of the parallel algorithm when executed in the virtual computer M1 and for P varying from 1 to 10 processors, the fastest ones being chosen in each case. There is a decrease in the processing time until P reaches 6; above that number it increases. From these results one concludes that the ideal number of processors to be used for the 64kb image is 6 processors, as obtained theoretically.

The second example is for an image of 144×144 pixels, 20.25 kb, where the value obtained for P , for the M1 virtual computer, was 4.07 processors.

Experimentally, as shown in Figure 7, the best performance is obtained for 4 processors. This result confirms again the validity of the theoretical model used.

5.2 Histogram algorithm

Histogram computation is also an extensively used algorithm in image processing, often as a preprocessing stage of more elaborated algorithms. Basically, the algorithm consists in counting the occurrences of each pixel level. The input is an image and the output is a vector, of integer values, with length equal to the number of values that a pixel can assume. For an 8 bit representation, the length is $2^8 = 256$.

Each processor computes a segment of the histogram requiring for that to collect that segment from the other processors. Therefore, the amount of data that is required to exchange is independent of the distribution used, although dependant on the number of processors used, since each processor has to receive from the other $P - 1$ processors the histogram segment that it is assigned to compute. As shown in Table 1, the amount of data to exchange, b , is given as a function of P . This changes the expression to compute P , the ideal number of processors, which now becomes:

$$T(n, P) = \frac{C_1 P}{S} + \frac{n}{P} \left(\frac{1}{W} + T_E \right) P + k_1 T_L + \left(\frac{1}{W} + T_E \right) \frac{N}{P} (P - 1) P + k_2 T_L (P - 1) P + \frac{k_3}{P} \left(\frac{1}{W} + T_E \right) P + k_4 T_L + \frac{n \beta^\alpha}{P S} \quad (12)$$

$$\begin{aligned} \frac{\partial T}{\partial P} &= \frac{C_1}{S} + \left(\frac{1}{W} + T_E \right) N + 2k_2 T_L P - k_2 T_L - \frac{S n \beta^\alpha}{P^2 S^2} = 0 \quad (13) \\ &= 2T_L k_2 P^3 + \left(\frac{C_1}{S} + N \left(\frac{1}{W} + T_E \right) - T_L k_2 \right) P^2 - \frac{n \beta^\alpha}{S} = 0 \end{aligned}$$

The equation is a polynomial in P , of degree 3. Assuming an 8 bit representation for image pixels, N assumes the value of 256. The value of n is the image size in kb. The network parameters assume the same values as before. Since the histogram vector can be sent in a single 1024 packet, k_2 equals 1. Replacing these values in the above expression, one gets:

$$10 \times 10^{-3} P^3 + \left(\frac{0.80}{80} + 256 \times 0.15 \times 10^{-6} - 0.5 \times 10^{-3} \right) P^2 - \frac{n \times 0.150}{129.7} = 0$$

Figure 8 shows the theoretical number of processors that will give the best speedup, for the cases when the master process runs in a 80 Mflops computer or an 161 Mflops one. The machine response time depends on the speed of the computer that runs the master process, since the constant C_1 is divided by its speed. For images of 64 kb and 256 kb the optimum value of P is, respectively, 2.4 and 4.5, for a 80 Mflops master computer. For a 161 Mflops master computer, the values are 3.1 and 5.4 processors.

Figure 9 shows the measures of the processing time made with the virtual machine M2 for images of 64 kb and 256 kb. As obtained theoretically the

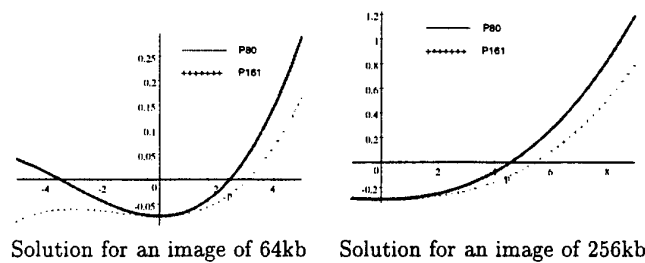


Fig. 8. Solution of the polynomial equation

optimum value practically found for P was, respectively, 2 and 4 when the master process runs in the 80 Mflops computer. When the master process is executed on the 161 Mflops computer, the optimum measured value is 2 and 5 processors. The value of $P = 5$ corresponds to the theoretical value for the 256 kb image; however, for the 64kb image, the theoretical value is 3 as opposed to the value 2 found practically. This discrepancy is in fact not very significant as the practical processing time found for 2 and 3 processors differs only slightly, as shown in Figure 9.

Additionally, the discrepancy can be justified by the fact that the processor running the master process also runs an instance of the slave process; as a consequence one can consider that the available capacity of this processor, as used by the slave process, is decreased; in fact for a lower capacity, the optimum theoretical number of processors would decrease.

Figure 9 also shows an important feature of the parallel virtual computer, namely the advantage of using the parallel version of the algorithm even when the user opts to launch a single slave process; in fact, when the user is logged on a slow machine, the serial version of the algorithm uses the same machine for the whole workload; if the parallel version is selected, then the master process is run on the slow machine, but the slave process is assigned to the fastest available computer, thus reducing the global processing time. This is confirmed by inspecting the measurements displayed in Figure 9 for the 80 Mflops curves: $P = 0$ corresponds to the serial algorithm and $P = 1$ corresponds to the parallel one. Notice also that when the user is logged on the fastest machine available (161 Mflops curves), the serial version is naturally faster, as the parallel one has communication overheads that are unnecessary in this situation of a single slave process.

5.3 Load Balancing

In a virtual parallel computer there are frequently machines of many different processing capacities, therefore the load distribution should be managed in order to assign more work to faster processors, so that all processors finish at about the same time. A test with machine M2 and the edge detection algorithm was

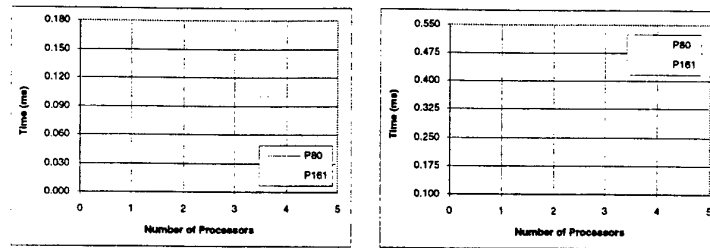


Fig. 9. Processing time of the histogram algorithm

carried out to verify the validity of the strategy suggested above. Since the computational load is evenly distributed over the domain it is straightforward to make the size of each row block proportional to the processor capacity. Figure 10 shows the response time and the time to process the edge detection algorithm kernel for each processor, for two situations: 3 and 4 processors working. Processors P1, P2, P3 and P4 have processing capacities of 161, 161, 105 and 91 Mflops respectively. The master process runs on processor P1.

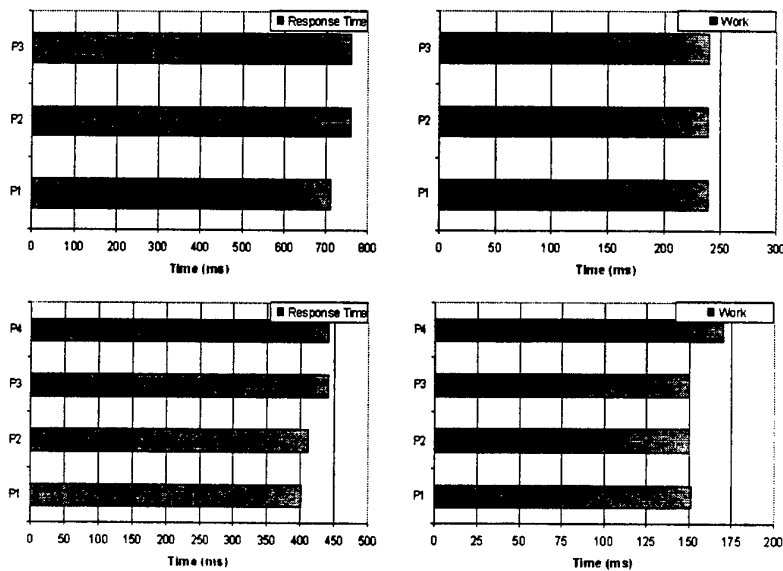


Fig. 10. Response time and work done by each processor

From Figure 10 it turns out that the load is well distributed, that is, a balanced workload was obtained even for largely different processing speeds of the Virtual Computer nodes. For algorithms where the load is not so evenly distributed over the domain, one has to have the additional work of blocking the domain in blocks of similar workload, for a static implementation of load balancing.

6 Conclusions

It was proved that for obtaining a good performance of the Parallel Virtual Computer, it is required to know the algorithm parameters, in order to compute the correct number of processors to use for its execution in the virtual computer. A methodology to obtain this number was presented, as well as some test results that proved its satisfactory accuracy.

The results suggest that a slow processor can be used to log on the user (and to run the master process), providing also some computation according to its capacity. This type of network allows a staged upgrade, since adding a fast computer to the network has a direct and positive impact on the global performance, no matter which processor is used to launch the algorithm.

References

1. Alves, A., Silva, L., Carreira, J., Silva, J.: WPVM: Parallel Computing for the People. In: Springer Verlag Lecture Notes in Computer Science: Proceedings of HPCN'95, High Performance Computing and Network Conference, Milan, Italy (1995) 582-587
2. Alves, A., Silva, J.: Evaluating the Performance of WPVM. *Byte* (May 1996)
3. Bader, D., JáJá, J.: Parallel Algorithms for Image Histogramming and Connected Components with an Experimental Study. CS-TR-3384, University of Maryland (December 94)
4. Al Geist et al: PVM 3 User's Guide and Reference Manual. Oak Ridge National Laboratory (1994)
5. Hwang, K.: Advanced Computer Architecture: Parallelism, Scalability, Programmability. McGraw-Hill (1993)
6. Pancake, C. M.: Is Parallelism for you? *IEEE Computational Science & Engineering* (Summer 1996) 18-37
7. Siegel, H. J.: Inside Parallel Computers: Trends in Interconnection Networks. *IEEE Computational Science and Engineering* (Fall 1996) 69-71
8. Steen, A. J.: Overview of Recent Supercomputers. Publication of the NCF, Nationale Computer Faciliteiten, The Netherlands (January 1997). <http://www.sara.nl/now/ncf>
9. Shen, J. and Castan, S.: An Optimal Linear Operator for Step Edge Detection. *CVGIP: Graphical Models and Image Processing*, Vol. 54. Number 2 (1992) 112-133
10. Spurgeon, C.: Ethernet Configuration Guidelines. Peer-to-Peer Communications, Inc. (Jan 96)

Behavioural Analysis Methodology oriented to Configuration of Parallel, Real-Time and Embedded Systems

F.J. Suárez, D.F. García

Universidad de Oviedo
Area de Arquitectura y Tecnología de Computadores
Campus de Viesques s/n, 33204 Gijón, Spain
E-mail: suarez@etsiig.uniovi.es

Abstract. This paper ¹ describes a methodology suitable for behavioural analysis of parallel real-time and embedded systems. The main goal of the methodology is to achieve a proper configuration of the system in order to fulfill the real-time constraints specified for it. The analysis is based on the measurement of a prototype of the system and is supported by a behavioural model. The main components of this model are known as "macro-activities", that is, the sequences of activities which are carried out in response to input events, causing the corresponding output events. This supposes a behavioural view in the analysis that complements the more usual structural and resource views. The methodology incorporates steps of diagnosis (evaluation of the causes of system behaviour) and configuration (planning of alternatives for design improvement after diagnosis). The experimental results of applying the methodology to the analysis of a well-known case study are also an important part of this paper.

1 Introduction

The motivation of this work comes from the lack of research works addressing jointly the three following aspects related to the analysis of systems: 1) Development of a methodology of system behavioural analysis; 2) Addressing the particular problems of real-time and embedded systems; and 3) Use of analysis metrics obtained from a event trace after system execution. Effectively, none of the research works among the more relevant ones in the area addresses together the three aspects. In [5] and [3] only the metric and real-time aspects are considered respectively. In [10], [2] and [9] methodological aspects in metric based analysis are shown. In [1] a set of metrics for real-time systems are used. Besides, regarding to the analysis metrics, this work defines metrics corresponding to the three possible system views [8], that is, behavioural, structural and resource views. Structural view is a static view of the system and provides information about its design. Resource view provides dynamic information about resource use and is, together with the structural view, the more usual view in system behavioural analysis. The last view, behavioural view, provides dynamic information about the temporal behaviour of the system in terms of sequences of activities along system execution.

¹ This research work has been supported by the ESPRIT HPC 8169 project ESCORT.

The organization of the rest of the paper is what follows: in point 2 the model used in the behavioural view is presented; in point 3 the main steps and aspects of the analysis methodology are shown; in point 4 the metrics that support the methodology and their utility are commented; in point 5 a well known case study is analyzed using the methodology; finally in point 6 the conclusions and future work are presented. Although the methodology can be applied to real-time systems in general, this work deals with parallel real-time embedded systems. hereafter referred to simply as *real-time systems* (RTS).

2 Behavioural model

To describe the temporal behaviour of RTS in terms of events, delays and actions, several approaches or behavioural models can be employed. A model based on an event-ordering graph has been selected as the behavioural model for this research work. The selection was made as a result of its simplicity, wide applicability and suitability for the proposed methodology. The model is composed of two main elements (see [8]): 1) Activities, which are represented by a sequence of three events: *Ready*, when the activity is ready to start; *Begin*, when it starts; and *End*, when it finishes; and 2) A set of precedence and synchronization relationships defined on the *ready* and *end* events, which establish partial ordering for a group of activities. These relationships give rise to the following kinds of activities: sequential activity (SEQ); activity of synchronization with other macro-activities (SYN); alternative or conditional execution activity (ALT); replicated activity (REP); and activity executed in parallel with others (PAR). These kinds of activities are represented in Figure 1.

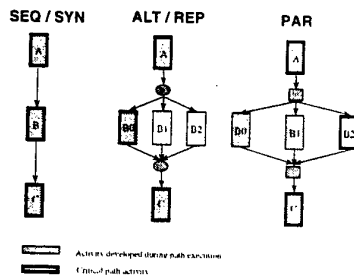


Fig. 1. Kinds of activities

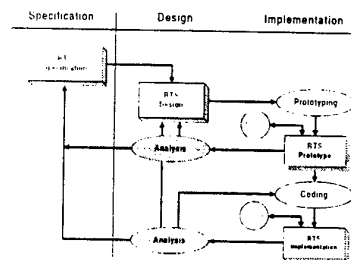


Fig. 2. RTS development cycle

In the model, three principal times are associated to each one of these activities to explain its temporal behaviour: Waiting Time (the time between the time-stamps of its *ready* and *begin* events), Service Time (the time between the time-stamps of its *begin* and *end* events) and Response Time (the sum of Waiting and Service Times). The model represents sequences of activities executed in response to the main input events which the RTS must deal with. These sequences, called *macro-activities* in the methodology,

are equivalent to the conventional *end-to-end tasks* found in related literature. Construction of the model implies an understanding of the functional and structural models provided by the RTS design methodology.

3 Analysis methodology

The goal of the methodology is the analysis of RTS behaviour from the early design phase to its final implementation. To achieve this, the methodology involves the construction of a synthetic prototype [4] of the RTS design, which is analyzed and refined until the validation of the timing requirements is achieved. It also serves as a skeleton for the implementation phase. In Figure 2 the integration of the methodology in the RTS development cycle is shown. Two refining cycles can be derived from the methodology; one working with early designs through prototypes of the RTS, and the other with the implementation of the RTS. The methodology approach can be resumed in the following 10 steps:

1. Prototyping of the initial RTS design under analysis.
2. Understanding of the sequences of activities to be executed as RTS responses to events (macro-activities), selecting the ones to be considered in the analysis, and establishing a **specification of behaviour** for them.
3. Instrumentation of the RTS software prototype to enable the **monitoring system** to obtain information about the behavior of macro-activities during the RTS execution.
4. Execution of the instrumented RTS under specific operational conditions (scenario) over a period of time long enough to obtain a representative event trace.
5. Checking of the fulfillment of the real-time constraints defined in the specification of behaviour for the RTS.
6. Development of a **multi-level analysis** in specific temporal analysis windows based on a set of **parameters and metrics** derived from the trace, and covering structural, behavioural and resource views.
7. Identification of critical macro-activities which do not fulfill their specifications of behaviour, and evaluation of the incidence of a set of possible causes of the behaviour observed in the RTS as a whole, the critical macro-activities and the critical activities within them (**diagnosis of temporal behaviour**).
8. Tuning of the system design according to the incidence of each cause of behaviour, establishing a suitable **configuration** of the RTS.
9. Repetition of the analysis cycle until a final prototype which permits timing validation is obtained.
10. Implementation of RTS design and repetition of the analysis cycle until its final implementation.

The main aspects of the methodology, highlighted above in bold face, are briefly described in the following points.

3.1 Specification of behaviour

The specification of behaviour of the RTS consists of specifications of the behaviour of each macro-activity. These specifications consider load characteristics and real-time

constraints in macro-activities. The most typical macro-activities in a RTS respond to periodic events and, consequently, are characterized by a periodic execution. These are called *periodic macro-activities*. Macro-activities responding to aperiodic events are called *aperiodic macro-activities*. The period of activation is the main load characteristic for periodic macro-activities, while the mean activation period and the typical deviation of it characterize aperiodic macro-activities. The real-time constraints considered for both periodic and aperiodic macro-activities are: the *absorption of productivity* and the *deadline (end-to-end deadline)*. The fulfillment of the first constraint implies capacity of the RTS to respond to all the input events produced during execution.

3.2 Monitoring system

The monitoring system, or simply the monitor, is highly dependent on the target system for which it is developed. In the context of this research work, a full software monitor for a multiprocessor based on T9000 transputers was developed [7]. The function of the monitor is to trace the occurrence of the most relevant software events during an application execution, and to store information related to them in a set of trace files. So, the functionality of the monitoring system consists of *run-time events* (communications, synchronization operations, I/O operations, etc.), *macro-activity events* (start) and *activity events* (ready, begin and end). The monitor is structured in three main components: a set of *distributed monitoring processes*, a collection of *instrumentation probes* spread over the application processes, and one *instrumentation data structure* per application process. In Figure 3 the instrumentation of one activity is shown. Finally, two steps are taken in order to improve the quality of measurement: precise synchronization of the system clocks (error < 10 microsec. with clock resolution = 1 microsec.) and reduction of the monitor intrusiveness (26-37 microsec./probe) to a minimum.

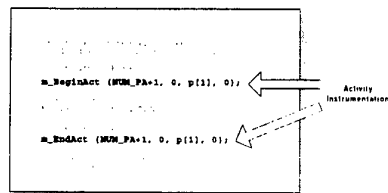


Fig. 3. Software instrumentation

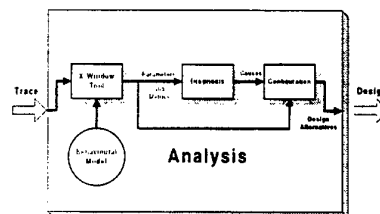


Fig. 4. Analysis approach

3.3 Multi-level analysis

This methodology allows the analysis of the system at three possible levels of abstraction. The first level considers the analysis of the RTS as a whole. The second level considers the analysis of each macro-activity of the RTS. Finally, the third level considers the analysis of each of the activities composing the macro-activities. This multi-level

character of the analysis permits a top-down approach, which is very useful in explaining the behaviour observed in the RTS. Figure 4 resumes the steps carried out during the analysis process. Starting from the event trace obtained by the monitor after RTS execution, an X-window tool permits the validation of the trace according to the behavioural model and generates the parameter and metric values. From these values, the diagnosis process is carried out, checking the fulfillment of the real-time constraints and obtaining the causes of behaviour of the RTS. Finally, with the causes of behaviour, parameters and metrics, the configuration process suggests alternatives for design improvement.

According to the multi-level analysis character described above, the methodology considers three possible analysis windows: RTS Window (a temporal window long enough to represent all the system behaviour characteristics for the scenario under analysis), Macro-activity Window (which corresponds to the longest response interval of a macro-activity within the RTS window) and Activity Window (which corresponds to the response interval of an activity within the macro-activity window). The RTS window is obtained from the *basic RTS period*, which corresponds to the Least Common Multiple (LCM) of all the periods specified for the periodic macro-activities, as seen in figure 5. The number of basic RTS periods in the RTS window is fixed considering factors such as the transient effect caused by pipelining, the statistic characteristics of the aperiodic macro-activities, and the variability of the response times in the macro-activities.

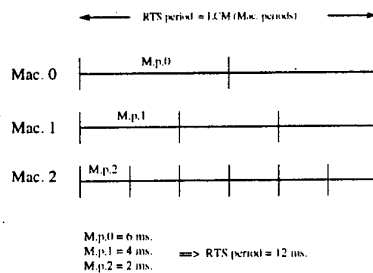


Fig. 5. Basic RTS period

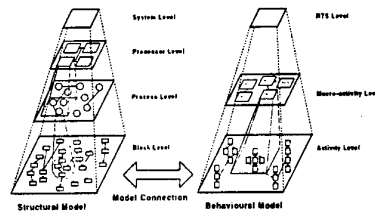


Fig. 6. Parameters in methodology

3.4 Parameters and metrics

Parameters resume all the known information about the RTS before execution. So, while they give only a static view of the system, they are necessary to determine the influence of the design components on the behaviour of the system. Four kinds of parameters are considered in the methodology: load parameters, parameters of the structural model, parameters of the behavioural model, and parameters of connection between models. Load parameters define the demands of service on the system from the environment, and reflect the load characteristics of macro-activities in the RTS behavioural model. The structural model defines the current design of the RTS and considers components on four levels or layers: the whole RTS, processors, processes (schedulable units) and

blocks. The blocks represent code sequences supporting the execution of activities. The behavioural model, on the other hand, considers components on three levels: the RTS as a whole, macro-activities and activities. Figure 6 shows the levels considered in each model and the relationships between them. The parameters of the structural and behavioural models represent mapping relationships between their components in the levels of the corresponding model. Finally, the parameters of connection between the models establish the mapping of activities in blocks, providing the basis with which to relate both models.

Metrics are the criteria to explain the behaviour observed in the system. They can be simple measurements obtained from the event trace, or relationships between the measurements and the parameters. Metrics provide information which feed the models corresponding to the behavioural and resource views. The magnitudes employed in the construction of the metrics corresponding to the behavioural view are the following: Initialization time of macro-activity (T_{iniM}); Response time of macro-activity (T_{resM}); Waiting time of macro-activity (T_{waiM}); Service time of macro-activity (T_{serM}); Number of executions of macro-activity (N_eM); Theoretic number of executions of macro-activity (N_{eTM}); Deadline of macro-activity (DM); Number of failures of deadline in macro-activity (N_{fDM}); Response time of activity (T_{resA}); Waiting time of activity (T_{waiA}); Service time of activity (T_{serA}); and Number of executions of activity (N_eA).

On the other hand, the new magnitudes employed in the construction of the metrics corresponding to the resource view are the following: Process use of RTS (U_{proS}); Processor use of RTS (U_{cpuS}); Set of processors use of RTS (U_{cpuS}); Time of processor use of macro-activity (T_{cpuM}); Communication time of macro-activity (T_{comM}); Process use of macro-activity (U_{proM}); Processor use of macro-activity (U_{cpuM}); Set of processors use of macro-activity (U_{cpuM}); Index of concurrence of macro-activity (I_{cM}); Time of processor use of activity (T_{cpuA}); Communication time of activity (T_{comA}); Process use of activity (U_{proA}); Processor use of activity (U_{cpuA}); Index of blocking of activity (I_{bA}); Index of parallelism of activity (I_{pA}); and Index of concurrence of macro-activity (I_{cA}).

The metrics used in this methodology can be classified according to the level of analysis in which they are applied. So, three different levels can be distinguished: RTS level metrics, macro-activity level metrics and activity level metrics. For a specific level of analysis and a specific view, the metrics can be calculated using all three analysis windows, that is the RTS window (W_{rts}), the macro-activity window (W_{mac}) and the activity window (W_{act}). Tables 1 and 2 show all the metrics at activity level corresponding to the behavioural and resource views, respectively. M and D prefixes in metrics refer to the mean and deviation values respectively.

The Index of blocking (I_b) helps to identify the cause of blocking time in an activity. This index compares the activity response time with the macro-activity period, in order to establish if the blocking time is caused by overlapping of macro-activity executions (when the sum of the service and communication times is greater than the macro-activity activation period). Therefore, index values over 1 indicate overlapping. The Index of parallelism (I_p) provides information about the level of concurrence of parallel activities (PAR activities) of the behavioural model, in a given execution. The

Activity Level		
Wrts	Wmac	Wact
MTwaiA/MTresA		TwaiA/TresA
MTresA/MTresM		TresA/TresM
DTresA/MTresA		
NeA/NeM		

Table 1. Behavioural view

Activity Level		
Wrts	Wmac	Wact
MTcpuA/MTserA		TcpuA/TserA
DTcpuA/MTcpuA		
MTcomA/MTwaiA		TcomA/TwaiA
MIbA		IbA, IpA, IcA
UproA	UproA	UproA
UcpuA	UcpuA	UcpuA

Table 2. Resource view

Index of concurrence (Ic) provides information about the level of concurrence of all the activities composing the macro-activity in a given execution.

3.5 Diagnosis of temporal behaviour

The stages to follow in the diagnosis of temporal behaviour are the following:

- *Stage 1: Diagnosis of the RTS.* In this first stage, global causes of behaviour of the set of macro-activities in the behavioural model are evaluated.
- *Stage 2: Identification of critical macro-activities.* The objective of this stage is the identification of macro-activities which do not fulfill one or more of the real-time constraints defined in the specification of behaviour.
- *Stage 3: Diagnosis of each critical macro-activity.* In this stage, the causes of behaviour which explain the response time of each critical macro-activity are found.
- *Stage 4: Identification of critical activities.* The objective of this stage is the identification of the most significant or critical activities (bottlenecks) in each critical macro-activity.
- *Stage 5: Diagnosis of each critical activity.* In this last stage, the causes of behaviour which explain the response time of each critical activity are found.

The support provided by parameters and metrics in the diagnosis is shown in figure 7. The figure resumes the parameters and metrics (including the analysis window considered for them) useful at each diagnosis stage.

When considering the causes of temporal behaviour in the diagnosis, the differences between the causes at activity level and the causes at macro-activity and RTS levels must be clearly established. At activity level, the response time of the critical activities must be explained, and three levels of diagnosis are considered. The first level of diagnosis evaluates the waiting and service times of each activity. Evaluation of communication and blocking times during waiting time, processing during service time and resource contention during both waiting and service times, correspond to the second level of diagnosis. Finally, the third level of diagnosis evaluates the specific causes of communication, processing, blocking or contention. Based on these three diagnosis levels, a set of causes of temporal behaviour of the critical activities can be established. Figure 8 represents the three levels of diagnosis. In table 3 all the causes considered, with a brief description of each, are detailed. The incidence of each cause of behaviour is evaluated

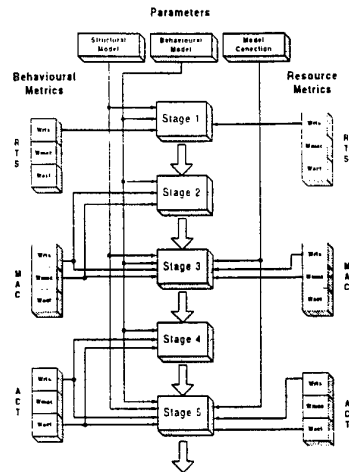


Fig. 7. Parameter and metric support in diagnosis

with a value in the range of 0–1. This value of incidence is the product of the partial incidence values obtained at each level of diagnosis.

At RTS level, the causes of behaviour of the RTS as a whole must be found, whereas at macro-activity level, the response time of critical macro-activities must be explained. To achieve those objectives, the aggregation of macro-activity and activity metrics are considered respectively. Therefore it is not possible to distinguish all the causes of behaviour considered at activity level. Only the first two levels of diagnosis are considered, and so the causes of behaviour are reduced to five. These causes are the following: **INI** (initialization time of macro-activities); **WAL.COM** (communication during waiting time); **WAL.BLO** (blocking during waiting time, contention included); **SER.CON** (contention during service time); and **SER.PRO** (processing during service time);

3.6 Configuration

Once the incidence of the causes of the behaviour of each critical activity composing the critical macro-activities has been established, those with high incidence will be considered, in order to tune the RTS design and establish its proper configuration. The goal of configuration can be either the fulfillment of the real-time constraints using the available resources, or the reduction of resources in the RTS design, while maintaining the fulfillment of the real-time constraints. The proper design alternatives for the causes of behaviour of critical activities within the critical macro-activities are the following: **Re-mapping of blocks on processes (RBL)**; **Re-mapping of processes on processors (RPR)**; **Change of process priority (CPR)**; **Segmentation of an activity (SEG)**; **Replication of an activity (REP)**; **Parallelization of an activity (PAR)**; **Balance of load in PAR activities (BAL)**; and **Optimization of block code (OPT)**.

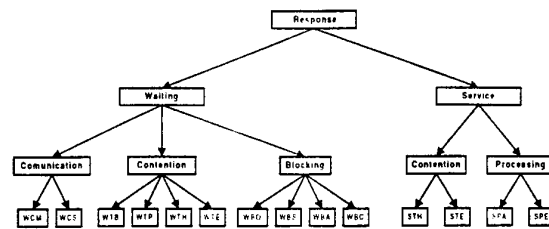


Fig. 8. Levels of diagnosis

Cause	Description
WCM	Communication with other activities of the same Macro-activity, during the Waiting time of the activity
WCS	Communication for Synchronization with other macro-activities during the Waiting time
WTB	ConTention of CPU due to competition with other activities supported by the same Block during the Waiting time
WTP	ConTention of CPU due to competition with activities supported by other blocks of the same Process
WTH	ConTention of CPU due to competition with Higher priority activities during the Waiting time
WTE	ConTention of CPU due to competition with activities of Equal priority during the Waiting time
WBO	Blocking during the Waiting time due to execution Overlapping in the macro-activity
WBS	Blocking during the Waiting time due to synchronization with other macro-activities
WBP	Blocking during the Waiting time of a PAR activity caused by the previous synchronization of other PAR activities
WBC	Blocking during the Waiting time due to other overheads associated with Communication
STH	ConTention of CPU due to competition with Higher priority activities during the Service time of the activity
STE	ConTention of CPU due to competition with activities of Equal priority during the Service time
SPP	Load imbalance of a PAR activity during the Processing part of the Service time
SPE	Execution of code during the Processing part of the Service time

Table 3. Causes of behaviour. Activity level

4 Case study

The case study considered here to show the use of the analysis methodology and described below has been widely studied in other papers, such as [6].

A Remote Speed Sensor (RSS) measures the speed of a number of motors, and reports them to a remote host computer. The speed of each motor is obtained by periodically reading a corresponding digital tacometer. The interval between speed readings (10-1000 ms.) for each motor is specified by the host computer. An Analogic to Digital Converter (ADC) with a set of multiplexed channels is used to measure the speed signal provided by tacometers coupled to the motors. The ADC accepts reading-requests in the form of motor numbers (integers in the range of 0-15). After a request has been received, the converter reads the speed of the motor, stores it in a hardware buffer, and generates an interruption. The converter can only read the speed of one motor at a time. The interval between readings for a given motor is specified in a control packet which is sent from the host computer to the RSS. The speed of a motor is reported to the host via a data packet. When a control packet is received from the host, it is checked for

validity. If the message is valid, an acknowledgment (ACK) is sent to the host. If it is not, a negative acknowledgment (NAK) is sent. When a data packet is sent, the RSS waits to receive either an ACK or a NAK from the host. If a NAK is received, or neither an ACK nor a NAK is received within half the reading interval, the message emission is marked as a failure.

4.1 Design structure

The design structure of the case study has four principal parts: the tacometer, the motors, the host interface and the host. Each of these parts is composed of one or more software processes, as seen in Figure 9. The tacometer process can access the speeds of all the motors and is constantly waiting to read requests. When a request is received, it reads the corresponding speed and sends the data to the motor process. Motor processes, one per motor, periodically send the reading-requests to the tacometer process. Once data is received, it is filtered and sent to the host. These processes have associated processes which inform them about new reading intervals requested from the host. The Host interface is implemented with four processes: *inport*, *outport*, *inmsg* and *outmsg*. *Inport* receives packets from the host. If the packet is a control packet, it is sent to *inmsg*. If it is an ACK or a NAK, it is sent to *outmsg*. *Outport* receives packets from *outmsg* and *inmsg* and sends them to the host. *Inmsg* receives control packets from *inport*. If the packet is not valid, it sends NAK to the host through *outport*. If it is valid, it sends ACK to the host through *outport* and the new interval to the motor process. *Outmsg* receives speeds from the motor processes and ACKs and NAKs from *inport*. All of them are sent to the host. The host has three processes. *Phost* is the main process and *phostin* and *phostout* are input and output processes for communication with their corresponding processes in the RSS host interface. The embedded system was implemented in a PARSYS SN9500 machine, a distributed memory parallel machine based on Transputers with 8 CPUs. In this machine the process communications are established through a virtual channel network. RSS processes were implemented over 4 CPUs. For simplicity in the case study, host processes were also implemented in the same machine using an extra CPU.

4.2 Behavioural model and real-time constraints

According to the methodology, the relevant action-reaction event couples which demand system response must first be identified. The action events are the periodic requests for speed-reading from the motor processes to the tacometer process. The reaction events are the arrivals at *outmsg* process of ACKs coming from the host in response to the emission of packets with speed data. A total of 16 kinds of events, one per motor, are considered. The macro-activities include all the activities executed from the speed-reading request to the ACK reception from the host. All 16 macro-activities considered in the analysis have the same structure, as shown in Figure 10, corresponding to the RTS behavioural model.

The real-time constraints of each macro-activity in the case study depend on the speed of the corresponding motor. The deadline of each macro-activity has the same

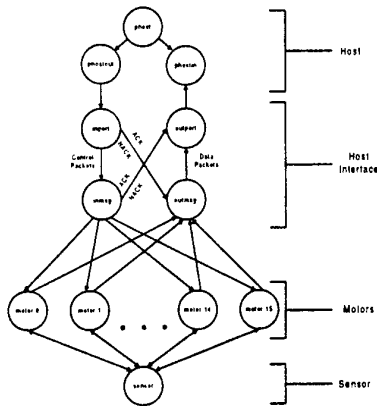


Fig. 9. Process Structure

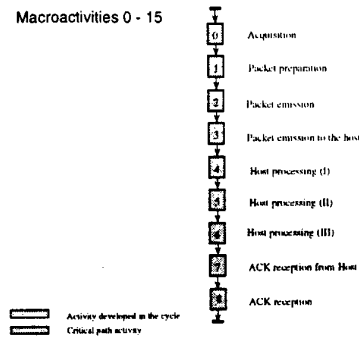


Fig. 10. Behavioural model

value as its period of execution, as shown in table 5, where *Mac.* refers to the macro-activity number, and times are given in milliseconds.

Mac.	Period	DM	Mac.	Period	DM
0	12	12	8	80	80
1	20	20	9	100	100
2	25	25	10	120	120
3	30	30	11	150	150
4	40	40	12	200	200
5	50	50	13	300	300
6	60	60	14	400	400
7	75	75	15	600	600

Table 4. Real-time constraints

CPU	Processes	Activities	Nblocks
T9000[0]	phost	{i,5} i=0-15	1
	phostin	{i,4} i=0-15	1
	phostout	{i,6} i=0-15	1
T9000[1]	inport	{i,7} i=0-15	1
	outport	{i,3} i=0-15	1
	inmsg	-	-
	outmsg	{i,2} {i,8} i=0-15	1
T9000[2]	motor[i]	{i,1} i=0,2...(even)	8
T9000[3]	motor[i]	{i,1} i=1,3...(odd)	8
T9000[4]	tacometer	{i,0} i=0-15	1

Table 5. Parameters

4.3 Analysis

Table 4 resumes the main parameters: parameters of the structural model, parameters of the behavioural model and parameters of connection between the models. It shows the mapping of processes in processors, the mapping of activities in the processes and the number of blocks giving support to the activities (*Nblocks*). Index *i* refers to macro-activities and activities {i,1} are the only ones supported in various independent blocks and processes.

Diagnosis

Stage 1. In Table 6 some of the RTS level metrics in the RTS analysis window are shown. A significant synchronization load from the relative value of Twai can be observed. A very low rate of deadline failures and a medium level use of the set of processors are also observed. Finally, Table 7 shows the incidence of the causes of behaviour for the RTS as a whole. Here, blocking is the most important cause of behaviour.

Metric	Value
M(MTiniM/MTresM)	0.025
M(MTwaiM/MTresM)	0.606
M(MTresM/DM)	0.213
M(NeM/NeTM)	1.000
M(NfDM/NeM)	0.001
M(MTcomM/MTwaiM)	0.183
M(MTcpuM/MTserM)	0.778
UcpusS	0.506

Table 6. RTS metrics

Cause	Incidence
INI	0.03
WAI.COM	0.11
WAI.BLO	0.50
SER.CON	0.08
SER.PRO	0.29

Table 7. RTS diagnosis

Stage 2. In Figure 11 the relative value of macro-activity response times with regard to their deadlines is shown. Only macro-activity 0 in its macro-activity window exceeds its deadline. The rest of metrics establish that the constraint of productivity absorption is fulfilled in all macro-activities and the deadline constraint is not fulfilled in macro-activity 0. So, macro-activity 0 is selected as a critical macro-activity.

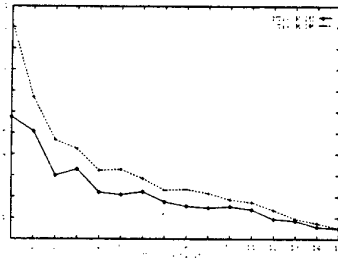


Fig. 11. Macro-activity level metrics

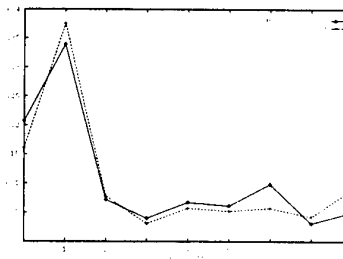


Fig. 12. Activity level metrics

Stage 3. Processing (0.52 incidence), blocking (0.29 incidence) and communication (0.11 incidence) result to be the principal causes of the behaviour in the macro-activity window for critical macro-activity 0.

Causes of behaviour														
Comp	WCM	WCS	WTB	WTP	WTH	WTE	WBO	WBS	WBP	WBC	STM	STI	SPP	SPE
WAI	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	-	-	-	-
COM	0.80	0.80	-	-	-	-	-	-	-	-	-	-	-	-
Mac	1.00	-	-	-	-	-	-	-	-	-	-	-	-	-
Syn	-	0.00	-	-	-	-	-	-	-	-	-	-	-	-
CON	-	-	0.00	0.00	0.00	1.00	-	-	-	-	-	-	-	-
Blo	-	-	0.00	-	-	-	-	-	-	-	-	-	-	-
Pro	-	-	-	0.00	-	-	-	-	-	-	-	-	-	-
Hpr	-	-	-	-	0.00	-	-	-	-	-	-	-	-	-
Epr	-	-	-	-	-	1.00	-	-	-	-	-	-	-	-
BLO	-	-	-	-	-	-	0.00	0.00	0.00	0.00	-	-	-	-
Ovr	-	-	-	-	-	-	0.00	-	-	-	-	-	-	-
Syn	-	-	-	-	-	-	-	0.00	-	-	-	-	-	-
Par	-	-	-	-	-	-	-	-	0.00	-	-	-	-	-
Com	-	-	-	-	-	-	-	-	-	0.00	-	-	-	-
SER	-	-	-	-	-	-	-	-	-	-	0.93	0.93	0.93	0.93
CON	-	-	-	-	-	-	-	-	-	-	0.51	0.51	-	-
Hpr	-	-	-	-	-	-	-	-	-	-	0.00	-	-	-
Epr	-	-	-	-	-	-	-	-	-	-	-	1.00	-	-
PRO	-	-	-	-	-	-	-	-	-	-	-	-	0.49	0.49
Par	-	-	-	-	-	-	-	-	-	-	-	-	0.00	-
Exe	-	-	-	-	-	-	-	-	-	-	-	-	-	1.00
TOTAL	0.06	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.47	0.00	0.46

Table 8. Diagnosis of behaviour for activity 1

Stage 4. Activity level metrics considered here correspond to activities composing the critical macro-activity 0. In figure 12 the relative value of each activity response time with respect to the macro-activity response time in the activity and RTS windows is shown. Activity 1 is seen to be the longest activity with nearly 40% incidence in the macro-activity response time. So, activity 1 is selected as a critical activity.

Stage 5. Table 8 shows the incidence of the causes of the behaviour in the activity window of critical activity 1. It also shows the partial incidence of each diagnosis level. The main causes of behaviour are: contention with other activities of equal priority (the corresponding activities [i,1] in the other macro-activities) supported by different processes; and execution of code. Both causes correspond to the service time.

Configuration. The parameters show that critical activity 1 of critical macro-activity 0 is placed in CPU2. Checking the use of this CPU by each macro-activity in the critical activity window, macro-activities 2 and 10 can be observed in strong competition with macro-activity 0. A new mapping with motor processes motor[2] and motor[10] in CPU2 eliminates deadline failures in all macro-activities.

5 Conclusions and future work

An analysis methodology applicable to configuration of parallel, real-time and embedded systems has been presented. The methodology is based on temporal behaviour analysis of the system and considers a behavioural model of the RTS composed of real-time *macro-activities* giving response to the input events. The methodology involves the construction of a synthetic prototype for the initial design of the RTS, which is refined until the final implementation. Configuration of the RTS is achieved previous diagnosis of its temporal behaviour, based on a set of parameters and metrics covering three complementary views of the system: behavioural view, structural view and resource view. A well known case study has been analyzed with the methodology and demonstrated its possibilities in configuration of RTS.

Future work has two main objectives. Firstly, to derive automatic rules for proper configuration of systems from the expertise gained with the use of the methodology. Secondly, to apply the methodology to real-time POSIX applications implemented with either parallel or distributed architectures.

References

1. Rolf Borgeest, Bernward Dimke, and Olav Hansen. A trace based performance evaluation tool for parallel real-time systems. *Parallel Computing*, 21(4):551–564, April 1995.
2. J.K. Hollingsworth. *Finding Bottlenecks in Large Scale Parallel Programs*. PhD thesis, University of Wisconsin-Madison, 1994.
3. O. Pasquier, J.P. Calvez, and V. Henault. A complete toolset for prototyping and validating multi-transputer applications. In Monique Becker, Luc Litzler, and Michel Trhel, editors, *Transputers'94. Advanced Research and Industrial Applications*, pages 71–86. IOS Press, 1994.
4. David A. Poplawsky. Synthetic models of distributed-memory parallel programs. *Journal of Parallel and Distributed Computing*, 12:423–426, 1991.
5. L. Schäfers, C. Scheidler, and O. Krämer-Fuhrmann. Trapper: A graphical programming environment for embedded mmd computers. In R Grebe et Al., editor, *Transputer Applications and Systems'93*, pages 1023–1034. IOS Press, 1993.
6. C.U. Smith. Software performance engineering: A case study including performance comparison with design alternatives. *IEEE Transactions on Software Engineering*, 19(7):120–141, July 1993.
7. F.J. Suárez, J. García, S. Grana, D. García, and P. de Miguel. A toolset for visualization and behavioural analysis of parallel real-time systems based on fast prototyping techniques. In *6th Euromicro Workshop on Parallel and Distributed Processing*. IEEE Computer Society, January 1998.
8. C.M. Woodside. A three-view model of performance engineering of concurrent software. *IEEE Transactions on Software Engineering*, 21(9):754–767, September 1995.
9. J.C. Yan and S.R. Sarukkai. Analyzing parallel program performance using normalized performance indices and trace transformation techniques. *Parallel Computing*, 22:1215–1237, 1996.
10. C. Yang and B.P. Miller. Performance measurement for parallel and distributed programs: a structured and automatic approach. *IEEE Transactions on Software Engineering*, 15(12):1615–1629, December 1989.

High Performance Computing for Image synthesis

Thierry Priol

IRISA, Campus de Beaulieu - 35042 Rennes Cedex - France
priolirisa.fr

Abstract. For more than 20 years, several research works have been carried out to design algorithms for image synthesis able to produce photorealistic images. To reach this level of perfection, it is necessary to use both a geometrical model which accurately represents an existing scene to be rendered and a light propagation model which simulates the light propagation into the environment. These two requirements imply the use of high performance computers which provide both a huge amount of memory to store the geometrical model and fast processing elements for the computation of the light propagation model. Moreover, parallel computing is the only available technology which satisfies these two requirements. Since 1985, several technology tracks have been investigated to design efficient parallel computers. This variety forced designers of parallel algorithms for image synthesis to study several strategies. This paper presents these different parallelisation strategies for the two well known computer graphics techniques: ray-tracing and radiosity.

Keywords: parallel rendering, ray-tracing, radiosity

1 Introduction

Since the beginning of the last decade, a lot of research works were made to design fast and efficient rendering algorithms for the producing of photorealistic images. Such efforts were aimed at both having more realistic light propagation models and at reducing the algorithm complexity. Such objectives were driven by the need to produce high quality images having several millions of polygons. Despite these efforts and the increasing performance of new microprocessors, computation times remains at unacceptable level. Using both a realistic light propagation model and an accurate geometrical model require huge computing resources both in term of computing power and memory. Only parallel computers can provide such resources to produce images in a reasonable time frame. For more than 10 years, the design of parallel computers was in constant evolution due to the availability of new technologies. During the last decade, most of the parallel computers were based on the distribution of memories, each processor having its own local memory with its own address space. Such Distributed Memory Parallel Computers (DMPC) have to be programmed using a communication

model based on the exchange of messages. Such machines were either SIMD (Single Instruction Multiple Data) or MIMD (Multiple Instruction Multiple Data). Among those machines, one can cite the latest machines CM-5, Paragon XP/S, Meiko CS-2, IBM SP-2. More recently, a new kind of parallel systems were available. Scalable Shared Memory Parallel Computers (SMPCs) are still based on distributed memories but provide a single address space. So that, parallel programming can be performed using shared variables instead of message passing. The latest SMPCs are HP/Convex Exemplar or SGI Origin 2000. Designing parallel rendering algorithms for such various machines is not a simple task since, to be efficient, algorithms have to take benefits of the specificities of each of these parallel machines. For instance, the availability of a single address space may simplify greatly the design of a parallel rendering algorithm. This paper aims at presenting different parallelisation strategies for two well known rendering techniques: ray-tracing and radiosity. These two techniques address two different problems when realistic images have to be generated. Ray-tracing is able to take into account direct light sources, transparency and specular effects. However, such technique does not take into account indirect lights coming from the objects belonging to the scene to be rendered. This problem is addressed by the radiosity technique which is able to simulate one of the most important form of illumination, the indirect ambient illumination provided by light reflected among the many diffuse surfaces that typically make up an environment. The paper is organised as follows. The next section gives some insights on parallelisation techniques. Section 3 gives an overview of techniques for the parallelisation of both the ray-tracing algorithms and the radiosity algorithms. Section 4 describes briefly a technique we designed to enhance data locality for a progressive radiosity algorithm. Conclusions are presented in section 5.

2 Parallelisation techniques

Using parallel computers require the parallelisation of the algorithm prior to their execution. Due to the different technology tracks followed by parallel computer designers, such parallel algorithms have to deal with different parallel programming paradigms (message-passing or shared variables). However, although the paradigms are different, designing efficient parallel algorithms require to pay attention to data locality and load-balancing issues. Exploiting data locality aims at reducing communication between processors. Such communication can be either message-passing when there are several disjoint address spaces or remote memory access when a global address space is provided. Data locality can be exploited using different ways.

A first approach consist in distributing data among processors followed by the distribution of computations. This later distribution has to be performed in such a way that computations assigned to a processor will access as much as possible local data which have been previously distributed. It consists in partitioning the data domain of the algorithm into sub-domains. Each of them is associated with a processor. Computations are assigned to the processor which owns the data

involved by these computations. If these latter generate new works that might require some other data not located in the processor, they are sent to relevant processors by means of messages or remote memory accesses. This approach is called *data oriented* parallelisation.

The second approach consist in distributing computations to processors in such a way that computations assigned to a processor will reused as much as possible the same data. Such approach requires a global address space since data have to be fetched and cached into a local processor. Such approach can be applied to the parallelisation of loops. Loops are analysed in order to discover dependencies. A set of tasks is then created, representing a subset of iterations. This approach is called *control oriented* parallelisation.

The last parallel approach, also called *systolic parallelisation*, breaks down the algorithm into a set of tasks, each one being associated with one processor. The data are then passed from processor to processor. A simplified form of systolic parallelisation is the well know pipelining technique.

For DMPCs, the physical distribution of processing elements makes *data oriented* parallelisation the natural way. However, to be efficient, such technique has to be applied to algorithms where the relationship between computation and data accesses is known. If such relationship is unknown, achieving a load balancing will be a tough problem to solve. For SMPCs, the availability of a global shared address space makes this task easier. When a processor has not enough computations to perform, it can synchronise with other processors to get more computations. However, such approach suffer by an increasing number of communications, since the idle processor will have to get data used by these new computations. A tradeoff has often to be found to get the maximum performance of the machine.

3 Parallel rendering

3.1 Ray-tracing

Principle The ray tracing algorithm is used in computer graphics for rendering high quality images. It is based on simple optical laws which take effects such as shading, reflection and refraction into account. It acts as a light probe, following light rays in the reverse direction. The basic operation consists in tracing a ray from an origin point towards a direction in order to evaluate a light contribution. Computing realistic images requires the evaluation of several million light contributions to a scene described by several hundred thousand objects. This large number of ray/object intersections makes ray tracing a very expensive method. Several attempts have been proposed to reduce this number. They are based on an object access data structure which allows a fast search for objects along a ray path. These data structures are based either on a tree of bounding boxes or on space subdivision.

Parallelisation strategies Ray tracing is intrinsically parallel since the evaluation of one pixel is independent of the others. The difficulty in exploiting this

parallelism is to simultaneously ensure that the load be balanced and that the database be distributed evenly among the memory of the processors. The parallelisation of such an algorithm raises a classical problem when using distributed parallel computers: how to ensure both data distribution and load balancing when no obvious relation between computation and data can be found? This problem can be illustrated by the following schematic ray tracing algorithm:

```

for i = 1, xpix do
  for j = 1, ypix do
    pixel[i, j] =  $\Sigma(\text{contrib}(\dots, \text{space}[f_x(\dots), f_y(\dots), f_z(\dots)], \dots))$ 
  done
done

```

The computation of one *pixel* is the accumulation of various light contributions *contrib()* depending on the lighting model. Their evaluations require the access to a database which models the scene to be rendered. In the ray tracing algorithm, the database *space* is both an object access data structure (space subdivision) and objects. The data accesses entail the evaluation of functions f_x , f_y and f_z . These functions are known only during the execution of the ray tracing algorithm and depend on the ray paths. Therefore, relationships between computation and data are unknown.

Parallelisation strategies explained in section 2 can be applied to the ray-tracing in the following manner. A *data oriented* parallelisation approach consists in distributed geometrical objects and their associated data structures (tree of extents or space subdivision) among the local memories of a DMPC. Each processor is assigned one part of the whole database. There are mainly two techniques for distributing the database, depending on the objects access data structure which is chosen. The first one partitions the scene according to a tree of extents while the second subdivides the scene extent into 3D regions (or voxels). Rays are communicated as soon as they leave out the region associated with one processor. From now on, this technique will be named **processing with ray dataflow**.

The *control oriented* parallelisation consist in distributing the two nested loops of the ray-tracing algorithms as shown previously. Such technique may apply to DMPCs but requires the duplication of the entire object data structure in the local memory of each processor. In that case, there is no dataflow between processors since each of them has the whole database. Pixels are distributed to processors using a master/slave approach. However, the limited size of the local memory associated with each processor of a DMPC prohibits its use for rendering complex scenes. A more realistic approach consists in emulating a shared memory when using a DMPC or to choose a SMPC which provides a single address space. The whole database is stored in the shared memory and accessed whenever it is needed. As said in section 2, such technique relies mainly of the exploitation of data locality. Ray-tracing has such property. Indeed, two rays shot from the observer through two adjacent pixels have a high probability of intersecting the same objects. This property is also true for all the rays

spawned from the two primary rays. Such property can be exploited to limit the number of remote accesses when computing pixels. Those algorithms use a scheme of **processing with object dataflow**. Both data oriented and control oriented approaches can be used simultaneously. Such hybrid approach has been investigated recently to provide a better load balance for the design of a scalable and efficient parallel implementation of ray-tracing. Concerning *systolic oriented* parallelisation, several studies have been carried out but the irregular nature of ray-tracing algorithms make such approach ineffective. Table 1 gives a list of references dealing with the parallelisation of ray-tracing depending on their parallelisation strategies.

Type of parallelism	Communication	Data Structure	References
Control	No dataflow		Nishimura et al. [29]
		Tree of extents	Bouville et al. [7] Naruse et al. [27]
	Object dataflow	Space subdivision	Green et al. [21, 20] Badouel et al. [4, 3] Keates et al. [24]
		Bounding volumes	Potmesil et al. [30]
Data	Ray dataflow	Space subdivision	Dippé et al. [16] Cleary et al. [12] Isler et al. [23] Nemoto et al. [28] Kobayashi et al. [25] Priol et al. [31] Caubet et al. [9]
			Salmon et al. [36] Caspary et al. [8]
		Tree of extents	
Hybrid	Object+Ray dataflow	Space subdivision	Reinhard et al. [34]

Table 1. Parallel ray tracing algorithms.

3.2 Radiosity

Principle Contrary to the ray-tracing technique, the radiosity method does not produce an image. It is a technique aiming at computing the indirect ambient illumination provided by inter-reflections of lights between diffuse objects. Such computations are independent of the view direction. Once such computations have been performed, the image of the scene is then computed by applying Gouraud shading or ray-tracing. The radiosity method assumes that all surfaces are perfectly diffuse, i.e. they reflect light with equal radiance in all directions. The surfaces are subdivided into planar patches for which the radiosity at each point is assumed to be constant. The radiosity computation can be represented by this equation:

$$B_i = E_i + \rho_i \sum_{j=1}^{j=N} F_{ji} B_j$$

where,

- B_i : Exitance of patch i (Radiosity) ;
- E_i : self-emitted radiosity of patch i ;
- ρ_i : reflectivity of patch i ;
- F_{ji} : form-factor giving the fraction of the energy leaving patch j that arrives at patch i ;
- N : number of patches.

The solution of this system is the patch radiosities which provide a discrete representation of the diffuse shading of the scene. In this equation, the most important component is the calculation of the form factor F_{ji} which gives the fraction of the energy leaving patch j that arrives at patch i . The computation of each form factor corresponds to the evaluation of an integral which represent the major computation bottleneck of the radiosity method. Form factor computations are carried out using several projection techniques such as the hemi-cube or the hemisphere [14]. Form-factors must be computed from every patch to every other patch resulting in memory and time complexities of $O(n^2)$. The very large memory required for the storage of these form-factors limits the radiosity algorithm practically. This difficulty was addressed by the progressive radiosity approach [13]. In the conventional radiosity approach, the system of radiosity equations is solved using Gauss-Siedel method. At each step the radiosity of a single patch is updated based on the current radiosities of all the patches. At each step, illumination from all other patches is gathered into a single receiving patch. Progressive radiosity method can be represented by the following schematic algorithm (N is the number of patches):

```

real Fi[N];                               /* column of form-factors */
real ΔRad;
real B[N];                                 /* array of radiosities */
real ΔB[N];                               /* array of delta radiosities */

for all patches  $i$  do                     /* Initialisation: delta radiosity =self-emittance */
    ΔB[i] =  $E_i$ ;
/* iterative resolution process */
while no convergence() do {               /* emission loop */
     $i = \text{patch-of-max-flux}()$  ;
    compute form-factors  $F_{i[j]}$ ;          /* form-factor loop */
    for all patches  $j$  do {                 /* update loop */
        ΔRad =  $\rho_j \Delta B[j] \times F_{i[j]} A_j / A_i$ ;
        ΔB[i] = ΔB[i] + ΔRad ;
        B[i] = B[i] + ΔRad ;
    }                                       /* end of update loop */
    ΔB[i] = 0.0;
}                                           /* end of emission loop */

```


At each step, the illumination due to a single patch is distributed to all other patches within the scene. Form factors can be computed either using a hemicube with classical projective rendering techniques or using a hemisphere with a ray-tracing technique. In the first steps, the light source patches are chosen to shoot their energy since the other patches will have received very little energy. The subsequent steps will select secondary sources, starting with those surfaces that receive the most light directly from the light sources, and so on. Each step increases the accuracy of the result that can be displayed. Useful images can thus be produced very early in the shooting process. Note that, at each step, only a column of the system matrix is calculated, avoiding thus the memory storage problem.

Parallelisation strategies The parallel radiosity algorithms proposed in the literature are difficult to classify, since numerous criteria can be considered: target architectures (SMPC, DMPC), type of parallelism (control oriented, data oriented, systolic), level at which the parallelism is exploited, form-factor calculation method, etc... The most important problem which arises when parallelising radiosity is the data access. Indeed, the form-factor calculation and the radiosity update require the access to the whole database. Another important point to be accounted for is the level at which the parallelism must be exploited in the algorithm. Indeed, a coarse grain reduces the cost entailed by managing the parallelism, but generates less parallelism than a fine grain. Moreover, data locality has to be exploited to keep as low as possible the amount of communications between processors. In the radiosity algorithm, choosing a coarse grain parallelism would not allow the exploitation of the data locality. Therefore, a trade-off between selecting a grain size and exploiting data locality has to be found. Several parallelisation strategies have been studied for both SMPCs and DMPCs. Table 1 gives references for some of these studies. Most of them focuses on the parallelisation of the progressive radiosity approach. In such approach, several levels of parallelism can be exploited. These levels of parallelism correspond to the three nested loops of the progressive radiosity algorithm as given above.

The first level consists in letting several patches to emit (or to shoot) their energy in parallel. Each processor is in charge of an emitter patch, and computes the form-factors between this patch and the others. Three cases can thus be considered. The first case is when all the processors are able to access all the patches' radiosities. In this case, each processor shoots the energy of its emitter patch and selects the next emitter patch. The second case occurs when each processor manages only a subset of radiosities. The form-factors computed by a processor are then sent to the other processors which update, in parallel, their own radiosities. The last case is when only one processor manages all the radiosities. The other ones take care of the calculation of a vector of form-factors (column of the linear system matrix) and send this vector to the master processor which updates all the radiosities and selects the next emitter patches. Note that such parallelisation changes the semantic of the sequential algorithm since the

emission order of patches varies with the number of processors available in the parallel machine. In addition, the selection of the emitter patches requires the access to all the patches' radiosities (global vision of the radiosities). Indeed, each processor selects the emitter patch among those it is responsible for, which slows down the convergence of the progressive radiosity algorithm.

The second level of parallelism consists in the computation of the form-factors between a given patch and the other ones by several processors.

The third level of parallelism corresponds to the computation of the delta form-factors in parallel. This level is strongly related to the classical projective techniques and z-buffering used for form-factor calculation.

Architecture	scene	Parallelism	References
SMPC	-	-	Baum et al. [5]
	shared	-	Renambot et al. [35]
DMPC	shared	control	Bouatouch et al. [6]
	distributed	data	Arnaldi et al. [1]
			Varshney et al. [38]
	distributed	control	Drucker et al. [17]
			Guitton et al. [22]
	distributed	control	Chalmers et al. [10]
	duplicated	control	Chen [11]
			Recker et al. [33]
	passed	systolic	Lepretre et al. [26]
			Purgathofer et al. [32]

Fig. 1. Classification of parallel radiosity algorithms

3.3 Discussion

As shown in the last section, parallelisation of the ray-tracing algorithm has been widely investigated and efficient strategies are now identified. Concerning radiosity computation, several parallelisation strategies have been studied. However, none of them really dealt with data locality. Therefore, solving the radiosity equation for complex scenes¹ in parallel cannot achieve good performance. In [39, 37, 2], several ideas have been proposed to deal with complex environments. They are mostly based on Divide and Conquer strategies: the complex environment is subdivided into several local environments where the radiosity computation is applied. In the following subsection, we present a new parallelisation strategy able to render complex scenes using a progressive radiosity approach as explained in section 3.2. It extends the work published in [2].

¹ scenes that have more than one million of polygons

4 Enhancing data locality in a progressive radiosity algorithm

The main goal of our work was to design a data domain decomposition well suited for the radiosity computation. Data domain decomposition aims at dividing data structure into sub-domains, which are associated to processors. Each processor performs its computation on its own sub-domain and send computations to other processors when it does not have the required data. Applied to the radiosity computation, our solution focuses on the ability to compute the radiosity on local environments instead of solving the problem for the whole environment. By splitting the problem into subproblems, using **Virtual Interface** and **Visibility Masks**, our technique is able to achieve better data locality than other standard solutions. This property is capital when using either a modern sequential computer to reduce data movement in the memory hierarchy or a multiprocessors to keep as low as possible communication between processors whatever the communication paradigm is: either message passing or shared memory.

4.1 Data domain decomposition

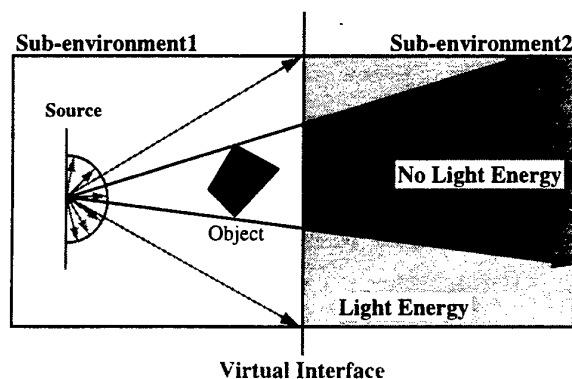


Fig. 2. Virtual Interface.

This section summarizes briefly the virtual interface that is the basic concept to perform a data domain decomposition. A more detailed description can be found in [1]. A virtual interface is a technique to split the environment (the scene bounding box) into local environments where the radiosity computation can be applied independently from other local environments (figure 2). It also addresses the energy transfer between local environments. When a source, located in a local environment, has to distribute its energy to other neighbouring local environments, its geometry and its emissivity has to be sent. We introduced

a new structure called the visibility mask, to the source (figure 3). The visibility mask stores in the source structure all the occlusions encountered during the processing in each local environment. With our virtual interface concept, the energy of each selected patch, called a source, is first distributed in its local environment. Then, its energy is propagated to other local environments. However, to propagate efficiently the energy of a given patch to another local environment, it is necessary to determine the visibility of the patch according to the current local environment: an object along the given direction may hide the source (figure 3). We introduced the visibility mask that is a sub-sampled hemisphere identical to the one involved in the computation of the form factors. To each pixel of the hemisphere, used for form factor computation, corresponds a boolean value in the visibility mask. The visibility mask allows the distribution of energy to local environments in a step by step basis. If the source belongs to the local environment, a visibility mask is created, otherwise the visibility mask already exists and will be updated during the processing of the source. Form factors are computed with the patches belonging to the local environment by casting rays from the center of the source through the hemisphere. If a ray hits an object in the local environment, the corresponding value in the visibility mask is set to false otherwise there is no modification. Afterwards, radiosities of local patches are updated during an iteration of the progressive radiosity algorithm. Finally, the source and its visibility mask are sent to the neighboring local environments to be processed later.

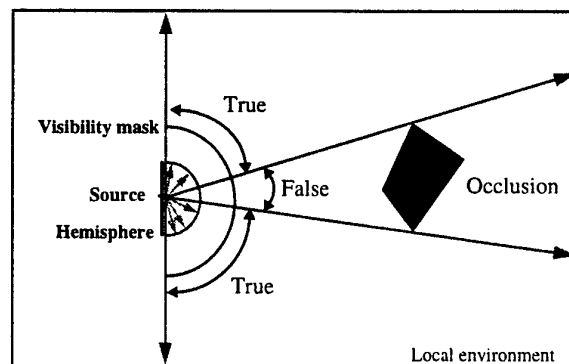


Fig. 3. Initialisation of the visibility mask for a local source.

4.2 Parallel algorithm

The data domain decomposition technique, as presented in the previous section, is well suited for distributed memory parallel computers. Indeed, a local environment can be associated with a processor performing the local radiosity computation. Energy transfer between local environments is performed using

message passing. The parallel algorithm running on each processor consists in three successive steps: an initialisation, a radiosity computation and a termination detection as shown in figure 4. These three steps are described in the following paragraphs.

```

void ComputeNode()
{
    Initialisation()
    do {
        do { /* Computing */
            Choose source among region and network queue;
            Shooting the source;
            Read all the sources arrived from network, put them in queue;
        } while(! local_convergence);
        TerminationDetection(done)
    } while ( not done);
}

```

Fig. 4. Algorithm running on a processor.

Each processor performs an initialisation step which consists in reading the local environment geometries that have been assigned to it. Once each processor has the description of its local environment geometries, it reads the scene database to extract polygons that belongs to its local environments. The last processing step consists in subdividing the local environment into cells using a regular spatial subdivision [18] in order to speedup the ray-tracing process when form factors are computed.

Once the initialisation has been performed, each processor selects an emitter patch which has the greater delta-radiosity from either its local environment or from a receive queue which contains patches and visibility masks that have been sent by other processors since the last patch selection. Each selected patch is associated with a visibility mask that represents the distribution of energy. Initially, when a processor selects a patch that belongs to its local environment, the visibility mask is set to true. As the form-factor calculation progress, using a ray-tracing technique, part of the visibility masks is set to false if the rays hit an object in the local environment. Once the energy has been shot in the local environment, it is necessary to determine if there is still some energy to be sent to the neighbouring local environments. A copy of the visibility mask is performed for each neighbouring local environment. A ray-tracing is then performed for all pixels which have a true value. Intersection with the considered plane is performed to determine if there is energy entering in the neighbouring local environment. If a ray hits the plane that separates the two local environments, the corresponding pixel of the visibility mask is left unchanged otherwise it is set to false. At the end of this process, if the copies of the visibility mask have still some pixel values to true, they are sent to processors which own the neighbouring

local environments together with some information related to the geometry and the photometry of the emitter patch.

Each processor performs its computation independently from the other processors. Therefore, it has no knowledge about the termination of the other compute nodes. Termination detection is carried out using two steps. The first step consist in deciding to stop the selection of a local patch if its energy is under a given threshold. The service node is in charge of collecting, on a regular basis, the sum of the delta radiosities of the local environments as well as the energy that is under transfer between compute nodes. Depending of this information, the service node can inform compute nodes to stop the selection of new local patches. Once the first step is reached, buffers that contain sources coming from other compute nodes have to be processed. To detect that all the buffers are empty, a termination detection is carried out using a distributed algorithm based on the circulation of a token [15]. Compute nodes are organized as a ring. A token, whose value is initially true, is sent through the ring. If a compute node has sent a patch to another node in the ring since the last visit, the value of the token is changed to false. It is then communicated to the next compute node. If the compute node, that initiates the sending of the token, received it later with a true value, it broadcasts a message to inform all the compute nodes that the radiosity computation has ended.

4.3 Results

Experiments were performed using an 56 processors Intel Paragon XP/S using three different scenes. The *Office* scene represents an office with tables, chairs and shelves. The scene contains two lights on the ceiling. It's an open scene with few occlusions. It is made of roughly six hundred polygons. After meshing, 7440 patches were obtained. The second scene is a set of 32 similar rooms. Four tables, four doors open onto next rooms and one light source compose a room. This is a symmetrical scene with many occlusions. This file comes from benchmarks scenes presented at the 5th *Eurographics workshop on rendering*. After meshing 17280 patches were obtained. The last scene is the biggest one. It represents five floors of a building without any furniture and with one thousand light sources. This scene represents the *Soda Hall Berkeley Computer Science* building [19]. After meshing, 71545 patches were generated.

Since the computing time of the sequential version depends on the number of local environments, we took a decomposition of the scene into 56 local environments in order to avoid super-linear speedup. As said previously, decomposition is a straightforward automatic process without optimisation to balance the load among the processors. Despite that no effort was spent to solve the load balancing problem, speedups were quite good comparing to other parallelisation strategies previously published. We got a speedup of 11 for the *Office* scene, 40 for the *Rooms* scene and 24 for the *Building* scene when using 56 processors.

Unfortunately, the Paragon XP/S we used for our first experiments did not have enough memory for handling complex scenes. Moreover, due to the lack of hardware monitoring available within the processor, it was impossible to study

the impact of our technique on the memory hierarchy. We performed several experiments using a 32 processors Silicon Graphics Origin 2000 having 4 Gbytes of physical memory. A complete description of this work has been published in [35]. Although this machine provides a global address space, we did not use it to access data. It has been used to emulate a message passing mechanism to exchange data between processors. Therefore, the parallel algorithm is quite similar to the one described in the previous section. We used two new scenes that have a larger number of polygons. The first, named *Csb*, represents the Soda Hall Building. The five floors are made of many furnished rooms, resulting in a scene of over 400.000 polygons. It's an occluded scene. The second scene, named *Parking*, represents an underground car park with accurate cars models. The scene is over 1.000.000 polygons. It is a regular and open scene. We use a straightforward decomposition algorithm that places virtual interfaces evenly along each axis.

The first experiment we did concerns the study of the impact of our technique to exploit efficiently the memory hierarchy when using one processor. We ran our algorithm using one processor. For that purpose, we designed a sequential algorithm able to process several local environments instead of only one [35]. For the two considered scenes, we subdivided the initial environment into 100 local environments for the *Parking* scene and 125 local environments for the *Csb* scene. A gain factor of 4.2 on the execution times can be achieved for the *Parking* scene with 100 sub-environments, and 5.5 for the *Csb* scene. The main gain is given by a reduction of memory overhead due to a dramatic reduction of secondary data cache access time up to a factor of 30 for the *Parking* scene and a factor of 11 for the *Csb* scene. With the reduction of the working set, we enhance data locality and make a better use of the L2 cache. Data locality reduces memory latency and allows the processor to issue more instructions per cycle, which is a great challenge on a superscalar processor. The overall performance goes from 10 *Mflops* to 28 *Mflops*.

The last experiment was performed using different number of processors. For this experiment, we subdivided the environment into 32 and 96 local environments. Since the number of local environments has an impact of the sequential time, as shown in the previous paragraph, we decided to compute the speedup using the sequential times obtained with the same number of local environments. This protocol aims at avoiding super-linear speedup due to the memory hierarchy. Using 32 processors, we obtained a speedup of 12 for the *Parking* and the *Csb* scenes when using 32 local environments. By increasing the number of local environments to 96, we increased the speedup to 21 for the *Parking* scene and 14 for the *Scene*. This increasing performance is mainly due to a better load balance between processors. Since there are several local environments assigned to a processor, a cyclic distribution of the local environments to the processors balance evenly the load.

5 Conclusion

As shown in this paper, parallelisation of rendering algorithms have been largely investigated for now more than ten years. Proposed solutions are both complex and often not independent from the underlying architecture. Even if such solutions have proven their efficiencies, parallel rendering algorithms are not widely used in production, especially for the production of movies based on image synthesis techniques. Such fact can be explained as follow. A movie is a set of image frames that can be computed in parallel. Such trivial exploitation of parallelism can be illustrated by the production of the *Toy Story* movie, which was made entirely with computer generated images. The making of such movie was performed using a network of dozens of workstations. Each of them was in charge of rendering one image frame that took an average of 1.23 hours. Therefore, one has to raise the following question: why do we have still to contribute to new parallelisation techniques for rendering algorithms? Three answers can be given to this cumbersome question. The first one is obvious, there are still a need to produce an image in the shortest time (for lighting simulation application). The second answer comes from the fact that both radiosity and ray-tracing techniques can be applied to other kinds of applications (wave propagation, sound simulation, ...). For such applications, it is an iterative process that consists in analysing the results of a simulation before starting another simulation with different parameters. In such cases, parallelisation of ray-tracing and radiosity algorithms is required to speedup the whole simulation process. The last answer comes from the resource management problem. Image frame parallelism is the most efficient technique if the geometric (objects) and photometric (textures) databases fit in memory. If such databases exceed the size of the physical memory, frequent accesses to the disk will slow down the computation times. By using several computers to put their resources (both processors and memory) together, a single image can be computed more efficiently. The challenge for the next decade will be, no doubt, the design of new parallel rendering algorithms capable of rendering large complex scenes having several millions of objects. Parallel computers have not always to be seen as a mean to reduce computing times but also as a mean to compute larger problem size which cannot afford by sequential computer.

Acknowledgments

I would like to thank B. Arnaldi, D. Badouel, K. Bouatouch, X. Pueyo and L. Renambot who contributed to this work.

References

1. Bruno Arnaldi, Thierry Priol, Luc Renambot, and Xavier Pueyo. Visibility masks for solving complex radiosity computations on multiprocessors. *Parallel Computing*, 23(7):887-897, July 1997.

2. Bruno Arnaldi, Xavier Pueyo, and Josep Vilaplana. On the Division of Environments by Virtual Walls for Radiosity Computation. In *Proc. of the Second Eurographics Workshop on Rendering*, pages 198-205, Barcelona, 1991. Springer-Verlag.
3. D. Badouel, K. Bouatouch, and T. Priol. Ray tracing on distributed memory parallel computers: Strategies for distributing computation and data. *IEEE Computer Graphics and Application*, 14(4):69-77, July 1994.
4. D. Badouel and T. Priol. An Efficient Parallel Ray Tracing Scheme for Highly Parallel Architectures. In *Eurographics Hardware Workshop*, Lausanne, Switzerland, September 1990.
5. Daniel R. Baum and James M. Winget. Real time radiosity through parallel processing and hardware acceleration. *ACM Workshop on interactive 3D Graphics*, pages 67-75, mar 1990.
6. K. Bouatouch and T. Priol. Data Management Scheme for Parallel Radiosity. *Computer-Aided Design*, 26(12):876-882, December 1994.
7. C. Bouville, R. Brusq, J.L. Dubois, and I. Marchal. Synthèse d'images par lancer de rayons: algorithmes et architecture. *ACTA ELECTRONICA*, 26(3-4):249-259, 1984.
8. E. Caspary and I.D. Scherson. A self balanced parallel ray tracing algorithm. In *Parallel Processing for Computer Vision and Display*, UK, january 1988. University of Leeds.
9. R. Caubet, Y. Duthen, and V. Gaildrat-Inguibert. Voxar: A tridimensional architecture for fast realistic image synthesis. In *Computer Graphics 1988 (Proceedings of CGI'88)*, pages 135-149, May 1988.
10. Alan G. Chalmers and Derek J. Paddon. Parallel processing of progressive refinement radiosity methods. In *Second Eurographics Workshop on Rendering*, volume 4, Barcelone, may 1991.
11. Shenchang Eric Chen. A progressive radiosity method and its implementation in a distributed processing environment. Master's thesis, Cornell University, 1989.
12. J.G. Cleary, B. Wyvill, G.M. Birtwistle, and R. Vatti. Multiprocessor ray tracing. *Computer Graphics Forum*, 5(1):3-12, March 1986.
13. Michael F. Cohen, Shenchang E. Chen, John R. Wallace, and Donald P. Greenberg. A progressive refinement approach to fast radiosity image generation. *SIGGRAPH'88 Conference proceedings*, pages 75-84, aug 1988.
14. Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press Professional, Boston, MA, 1993.
15. E.W. Dijkstra, W.H.J. Feijen, and A.J.M. Van Gasteren. Derivation of a Termination Detection Algorithm for Distributed Computation. *Inf. Proc. Letters*, 16:217-219, June 1983.
16. M. Dippé and J. Swensen. An adaptative subdivision algorithm and parallel architecture for realistic image synthesis. In *SIGGRAPH'84*, pages 149-157, New York, 1984.
17. Steven M. Drucker and Peter Schröder. Fast radiosity using a data parallel architecture. *Third Eurographics Workshop on rendering*, 1992.
18. A. Fujimoto, T. Tanaka, and K. Iawata. ARTS : Accelerated Ray Tracing System. *IEEE Computer Graphics and Applications*, 6(4):16-26, April 1986.
19. T. Funkhouser, S. Teller, and D. Khorramabadi. The UC Berkeley System for Interactive Visualization of Large Architectural Models. *Presence, Teleoperators and Virtual environments*, 5(1):13-44, Winter 1996.
20. S. Green. *Parallel Processing for Computer Graphics*. MIT Press, 1991.

21. S.A. Green and D.J. Paddon. A highly flexible multiprocessor solution for ray tracing. *The Visual Computer*, 5(6):62-73, March 1990.
22. Pascal Guitton, Jean Roman, and Gilles Subrenat. Implementation Results and Analysis of a Parallel Progressive Radiosity. In *IEEE/ACM 1995 Parallel Rendering Symposium (PRS '95)*, pages 31-38,101, Atlanta, Georgia, October 1995.
23. V. İşler, C. Aykanat, and B. Ozgüç. Subdivision of 3d space based on the graph partitioning for parallel ray tracing. In *2nd Eurographics Workshop on Rendering*. Polytechnic University of Catalogna, May 1991.
24. M. J. Keates and R. J. Hubbard. Interactive ray tracing on a virtual shared-memory parallel computer. *Computer Graphics Forum*, 14(4):189-202, October 1995.
25. H. Kobayashi, T. Nakamura, and Y. Shigei. A strategy for mapping parallel ray-tracing into a hypercube multiprocessor system. In *Computer Graphics International'88*, pages 160-169. Computer Graphics Society, May 1988.
26. Eric Lepretre, Christophe Renaud, and Michel Meriaux. La radiosit  sur tranputers. *La lettre du tranputer et des calculateurs distribu s*, pages 49-66, dec 1991.
27. T. Naruse, M. Yoshida, T. Takahashi, and S. Naito. Sight : A dedicated computer graphics machine. *Computer Graphics Forum*, 6(4):327-334, 1987.
28. K. Nemoto and T. Omachi. An adaptative subdivision by sliding boundary surfaces for fast ray tracing. In *Graphics Interface'86*, pages 43-48, May 1986.
29. H. Nishimura, H. Ohno, T. Kawata, I. Shirakawa, and K. Omura. Links-1: A parallel pipelined multimicrocomputer system for image creation. In *Proc. of the 10th Symp. on Computer Architecture*, pages 387-394, 1983.
30. M. Potmesil and E.M. Hoffert. The pixel machine : A parallel image computer. In *SIGGRAPH'89*, Boston, 1989. ACM.
31. T. Priol and K. Bouatouch. Static load balancing for a parallel ray tracing on a mimd hypercube. *The Visual Computer*, 5:109-119, March 1989.
32. Werner Purgathofer and Michael Zeiller. Fast radiosity by parallelization. *Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, pages 173-184, jun 1990.
33. Rodney J. Recker, David W. George, and Donald P. Greenberg. Acceleration techniques for progressive refinement techniques. *ACM Workshop on Interactive 3D graphics*, pages 59-66, mar 1990.
34. Erik Reinhard and Frederik W. Jansen. Rendering large scenes using parallel ray tracing. *Parallel Computing*, 23(7):873-885, July 1997.
35. Luc Renambot, Bruno Arnaldi, Thierry Priol, and Xavier Pueyo. Towards efficient parallel radiosity for dsm-based parallel computers using virtual interfaces. In *Proceedings of the Third Parallel Rendering Symposium (PRS '97)*, Phoenix, AZ, October 1997. IEEE Computer Society.
36. John Salmon and Jeff Goldsmith. A hypercube ray-tracer. *The 3rd Conference on Hypercube Concurrent Computers and Applications*, 2, Applications:1194-1206, January 1988.
37. R. van Liere. Divide and Conquer Radiosity. In *Proc. of the Second Eurographics Workshop on Rendering*, pages 191-197, Barcelona, 1991. Springer-Verlag.
38. Amitabh Varshney and Jan F. Prins. An environment projection approach to radiosity for mesh-connected computers. *Third Eurographics Workshop on rendering*, 1992.
39. Hau Xu, Qun-Sheng Peng, and You-Dong Liang. Accelerated Radiosity Method for Complex Environments. In *Eurographics '89*, pages 51-61. Elsevier Science Publishers, Amsterdam, September 1989.

Modeling snow transport by wind A Cellular Automata

Alexandre Masselot and Bastien Chopard

CUI, University of Geneva
CH-1221 Geneva 4, Switzerland
alexandre.masselot/bastien.chopard@cui.unige.ch

Keywords: lattice Boltzmann model, parallel computations,
snow transport, hydrodynamics

Abstract

We propose a numerical model of snow particles transported by wind and deposited around an obstacle. We adopted a lattice Boltzmann approach to model the fluid (wind) with a BGK subgrid technique and we add solid particles (snow) moving on the same lattice. This problem shows how lattice methods (where only the essential microscopic rules of a phenomenon are extracted) can solve complex situations where classical numerical models (CFD codes with moving boundaries, erosion/deposition...) may be inadequate. Moreover, parallel computers are naturally suited to this class of problems and their architecture is a powerful investigation tool. Our 2D model has been tested with a wide range of field experiments.

1 Introduction

Snow transport by wind strongly influences many human activities. Examples are given by a pass road buried under a snowdrift or the creation of wind slabs and cornices which dramatically increase the avalanche danger above a road or a ski trail. To face this problem, one can adopt an active strategy: Building obstacles influences the deposition pattern and, when located windward a road, the obstacle may store snow and reduce the drift. Therefore, as the placing and shape of such obstacles are not trivial, the field expert could be helped by results from a numerical model.

Predicting how snow deposits and gets eroded under the action of wind requires in principle to solve turbulent Navier-Stokes equation with particles in suspension and dynamically changing boundary conditions. This is hardly tractable. We propose a new numerical approach to simulate snow deposition, free of the above complications and based on the lattice Boltzmann method. We model the phenomena at a "microscopic" level of description and consider very intuitive basic mechanisms. Our approach is tested with a wide range of situations and provides an unified view of the various processes involved. The same model could also be extended to simulate sand dunes formation or sedimentation problems.

Snow transport by wind is still a domain where little understanding has been achieved. Different patterns of deposit occur when looking at different scales (from the very small ripples (a few centimeters) up to 15 meters high cornices in high mountains). This phenomenon are a rough test for a numerical model.

Phenomenologically, snow transport (i.e erosion and deposition) has been divided in three main processes, each corresponding to a different scale:

- *Creeping*: particles are "rolling" on the surface or making very little jumps.
- *Saltation*: in the first half meter above the surface, snow particles have been observed to be ejected vertically and follow a ballistic trajectory [9].
- *Suspension*: it accounts for transport over larger scales (often seen as white smoke by mountains crests).

Dealing with microscopic interactions, the cellular automata approach proposes a unified view of such a complex phenomena. We will present here the mechanism of our model and compared its results with field experiments.

2 Modeling reality

To predict snow transport and model the snow erosion/deposition areas, both wind tunnel experiments (with fairly good accuracy over a mountain pass area [1]) and numerical computations have been investigated. The numerical approach can be split in two kinds: statistical methods based on comparisons with recorded data [17] (they can be accurate on geometrically simple situations), and direct techniques based on standard computational fluid dynamics (CFD) [16].

The latter approach is technically quite difficult: to compute the wind pattern, one has to solve the turbulent Navier-Stokes equation with dynamically

changing boundary conditions (to account for the evolution of the deposition layer). Creeping, saltation and suspension of snow are included separately [12] through dedicated equations and most of the time, due to the complexity of the model, one or two of these processes are neglected.

In this report we propose a new and radically different numerical approach to simulating snow deposition, free of the above complications and based on a cellular automata and lattice Boltzmann method. Instead of solving a differential equation, we propose a *model* of the phenomena. We consider a "microscopic" description, involving snow and wind "particles" and the essential basic interactions between them.

The cellular automaton approach exploits the fact that several levels of reality exist in physics [8]. On the one hand, there is the macroscopic level where phenomena are expressed in terms of rather abstract mathematical objects such as differential equations. On the other hand, there is the microscopic level of description where the interactions between the basic constituents are considered. An important results of statistical mechanics is that the macroscopic level of description depends very little on the details of the microscopic interactions. Rather, it depends on conservation laws and symmetries (most fluids obey the same equations of motion, though the molecular interactions differ). One can use this property to build a fictitious universe in which the microscopic interactions are particularly simple to simulate on a computer and whose macroscopic behavior is identical to that of the real system [6]. Moreover, since the evolution rule is local and identical for every lattice cell, massively parallel computers are naturally designed to run such models.

3 The cellular automata approach

The first major step in this direction was the so-called FHP lattice gas model, a two-dimensional cellular automata fluid proposed by Frisch, Hasslacher and Pommeau [5]. As shown in figure 1, the fluid is modeled as a large population of discrete particles moving synchronously, according to discrete time steps, along the links of a regular lattice and changing their direction when bouncing into each other. Cellular automata fluids are typically described by occupation numbers $n_i(\mathbf{r}, t) \in \{0, 1\}$ indicating the absence or presence of a particle entering site \mathbf{r} at time t , with a unit velocity \mathbf{v}_i pointing along lattice direction i (for instance, directions are labeled counterclockwise, 0 designates East). The equation of motion reads

$$n_i(\mathbf{r} + \tau \mathbf{v}_i, t + \tau) = n_i(\mathbf{r}, t) + \Omega_i(n(\mathbf{r}, t)) \quad (1)$$

where τ is the time step and Ω_i the collision term defined as a nonlinear combination of the n_i and expressing the balance of particle in direction i after the collision takes place. The collision rule is tailored so that *mass* (defined as $\sum_i n_i$) and *momentum* ($\sum_i n_i \mathbf{v}_i$) are locally conserved by the interaction. Some lattice sites can be turned into solid (representing either deposited snow

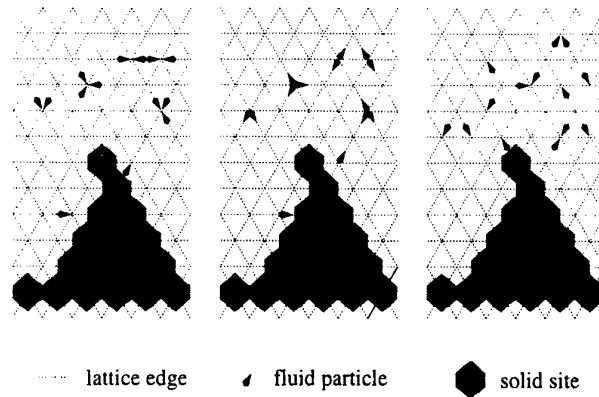


Fig. 1. Evolution of fluid particles on a hexagonal lattice. The first image shows incoming particles at time t , on each sites, the second one the distribution after the collision step, and the third one their new positions at time $t + \tau$. The lattice displayed here is hexagonal (as in the FHP model), but the one used for our simulations is square, where each cell has eight neighbors (E, NE, N, NW...).

or ground). Incoming particles on solid sites will bounce-back, thus modeling a no-slip boundary condition.

These types of models, despite their artifacts and limitations, capture the main essence of fluid dynamics, in the sense that the local average density ρ and velocity field \mathbf{u} :

$$\rho = \langle \sum_i n_i \rangle \quad \mathbf{u} = (1/\rho) \langle \sum_i n_i \mathbf{v}_i \rangle \quad (2)$$

obey, within certain limits, the Navier-Stokes equation with a built-in viscosity and pressure term. As explained in section 5, these algorithm are also very efficiently implemented on massively parallel computers, thus allowing an interactive modeling.

Much progress has been made simulating three dimensional fluids and remedying some of the earlier deficiencies (*e.g.* spurious invariants, statistical noise). A key improvement was to simulate the probability of a particle's presence $f_i = \langle n_i \rangle$ of a particle rather than the particle itself. This gives much more flexibility to define the collision rule. The so-called lattice BGK models [13] have no statistical noise and, above all, contain a free relaxation parameter $\tau_r > 1/2$ to tune the viscosity $\nu = (1/6)[2\tau_r - 1]$ (a review of this evolution is summarized by Qian [14]). The dynamics are similar to the FHP model and become

$$f_i(\mathbf{r} + \tau \mathbf{v}_i, t + \tau) - f_i(\mathbf{r}, t) = \frac{1}{\tau_r} [f_i^{(0)} - f_i] \quad (3)$$

where $f_i^{(0)}$ is the local equilibrium distribution depending on the local velocity field \mathbf{u} and particle density ρ :

$$f_i^{(0)} = a_i \rho + \frac{b_i}{v^2} \rho \mathbf{v}_i \cdot \mathbf{u} + \rho e_i \frac{u^2}{v^2} + \rho \frac{h_i}{v^4} v_{i\alpha} v_{i\beta} u_\alpha u_\beta \quad (4)$$

To lower the viscosity of the fluid, and therefore increase the Reynolds number of the flow, one can either decrease τ toward $1/2$ or increase the resolution of the lattice (at the cost of computer CPU time and memory). The first technique leads to numerical instabilities and the second one is limited by the power of available computers. To cope with such problems, a subgrid approach can be applied [7] following the large eddy simulations (LES) ideas: it takes into account eddies at unresolved scales (as the lattice spacing is finite) through their influence on larger ones. This influence can be included by adjusting dynamically and locally the relaxation time τ_r according to the magnitude of the local strain-rate tensor

$$\tau_r = \tau_0 + C [\partial_\beta u_\alpha + \partial_\alpha u_\beta]^2 \quad (5)$$

where C is a model constant and τ_0 the equilibrium relaxation time. The indices α and β label spatial coordinates and summations over repeated indices are assumed. Having such an effective relaxation time (and thus an effective viscosity) allows one to simulate high Reynolds number flows ($\sim 10^6$) by tuning C [7]. The effect of C is to adjust the resolution scale of the flow. A small C is appropriate to describe small developed eddies. Thus, in order to produce the correct logarithm velocity profile [19], which requires a finer resolution near the ground, C is considered as a function of the height [11]. It decreases linearly from a value C_∞ to 0 as one approaches the ground level.

The present model is far more elaborate than the original FHP but it keeps its main advantages: one deals directly with fluid "particles," the algorithm is simple, local and runs well on massively parallel computers.

4 Adding snow particles

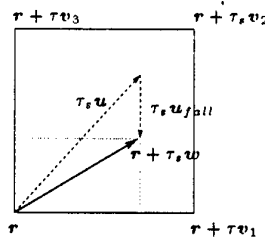
Virtual snow particles can be added on top of such a lattice BGK model for wind with simple erosion and deposition mechanisms. Snow particles can either be injected into the simulation (snow fall) or eroded from the ground, then deposited and transported according to the combined effect of gravity and wind direction.

It would be far too ambitious (and against our working hypotheses) to include all the complication of the erosion/transport phenomenon (exact number of snow particles, their geometrical structures, the correct inter-particle cohesion, and the evolution of the chemical properties and the temperature, etc.). Following the philosophy of statistical mechanics and the cellular automata approach, we restrict ourselves to what we identify as the most significant processes and express them in simple rules. The macroscopic behavior will be captured provided that these ingredients are indeed the relevant ones and that enough scale

separation exist between the different levels of description [2]. For the time being, no mechanism is included to compute the strain inside the snow deposit and, therefore, no realistic cornices can be modeled. The rules we consider are:

4.1 Transport

An arbitrary number of snow grains may reside at each lattice site. During the updating step, flakes are synchronously spread to the nearest neighboring sites following the wind velocity field \mathbf{u} and the gravity (through the falling speed \mathbf{u}_{fall}). As a solid particle moving from \mathbf{r} to $\mathbf{r} + \tau(\mathbf{u} + \mathbf{u}_{fall})$ would not remain on the (square) lattice, we define a randomized algorithm where only the averaged trajectories are exact:



In the case above, the grain would move eastwards with probability $p_{east} = \min(1, u_x/|v_0|)$ and northwards with $p_{north} = \min(1, u_y/|v_2|)$. Finally a particle will jump to neighbor 1 (*i.e.* $\mathbf{r} + \mathbf{v}_1$) with probability $p_1 = p_{east}p_{north}$, to 0 with $p_0 = p_{east}(1 - p_{north})$ and to 2 with $p_2 = p_{north}(1 - p_{east})$; it will therefore remain on \mathbf{r} with probability $p_{rest} = (1 - p_{east})(1 - p_{north})$. If $N = \sum N_i$ is large enough, one can approximate such a binomial scattering (among the neighbors 0, 1 and 2) with a Gaussian random distribution.

This algorithm obviously ensures that the average motion satisfies the local wind speed and gravity (*i.e.* the falling speed). Moreover, no effort is made to split the transport into creeping, saltation or suspension since a particle is not aware of its elevation above the snow ground level.

4.2 Deposition

Lattice sites can be either solid (original landscape or deposited snow) or free (air). A free site may become solid if enough virtual particles land on it. Snow particles on a free site may "freeze" if the neighboring site to which they want to jump is a solid site. When the number of frozen particles on a lattice site exceeds some (empirically) pre-assigned threshold, it becomes solid and subsequent incoming wind particles will bounce back. This threshold gives a way to assign some size to the snow particle, and must be calibrated in regard to the real size of a simulation cell. To model small phenomena, such as ripples (fig. 4), this threshold is lower (≈ 10 part.) than for the deposit around a mountain crest (≈ 100 part., fig. 2). When a site solidifies the wind particles that may be present get trapped until erosion frees them again.

4.3 Erosion

Deposited particles may be eroded under certain conditions. The erosion rate is not trivial. It is related to the wind speed above the solid site, the concentration of snow being transported, the saturation concentration, and the efficiency of the transport [18]. In our model, erosion means that snow particles are ejected, with a (low) probability p_{erod} , toward the upper neighboring site. Then, following the transport algorithm, such a particle will fly away if the wind velocity field is locally strong enough or fall back to its initial location. The quantity p_{erod} cannot be easily related to physical parameters, so it must be empirically calibrated: on the one hand, if p_{erod} is too high, no particle will remain at the same place for a long enough time and therefore no deposit can be built, but on the other hand, if it is too low, particles won't take off and nothing will happen before a long simulation time. This erosion rule seems trivial in view of the complexity of the problem, but the generation of ripples patterns at small space-scale shows that the rule may have caught a major point of the real erosion phenomenon.

5 Implementation and Performance

Both wind (equations (2) to (5)) and particle evolution computations (sections 4.1 to 4.3) are local and need regular communications (a cell communicates only with its nearest neighbors). The intrinsic simplicity of the model leads to a much simpler and compact programming (in CM Fortran, an early mix of Fortran 77 and High Performance Fortran dedicated to SIMD) compared to classic heavy CFD codes, solving the Navier Stokes differential equations.

As massively parallel computers are very well suited to lattice Boltzmann models, all our 2D snow deposits results have been performed on a Connexion Machine CM-200 with 8192 1-bit processors and 256 FPU. Some 3D fluid simulations have been achieved on an IBM SP2 (14 RS6000 processors, communicating through MPI).

5.1 Implementation

The simulation domain is mapped on a square lattice, thus 2D arrays naturally offer an efficient data structure. On every cell, we need (i) 9 floats for the wind densities (eight neighbors plus a rest quantity), (ii) 9 integers for the snow particles and (iii) 9 bits to get a local map of the surrounding solid cells.

The program execution can be summarized by:

1. Initialization
2. for every lattice site
 - 2.a. update the local map (9 bits) of the surroundings (as some neighboring cells may have frozen or been freed)
 - 2.b. compute the wind density and velocity according to equations (2)
 - 2.c. compute the snow particles evolution under

- transport (4.1), deposition (4.2) and erosion (4.3)
 - 2.d. a new wind particles distribution following eq. (3)
 - 2.e. propagate snow and wind densities according to their direction of motion
3. loop

To recover a realistic situation, a chronology of events (change of some parameters at given time steps) must be pre-defined. By example, one should (i) let the wind get established over the simulation domain, (ii) introduce snow particles and (iii) in some simulations, stop the particles injection and wait for a steady state deposit.

Moreover, to model snow transport by wind, many scenarios have had to be tested, leading to the fastidious exploration of a very large parameter space. Fortunately, short computing times (below one hour) allow the use of an interactive user-friendly tool, and the on-line tuning of the parameters in order to investigate relevant simulations.

5.2 Performances

A standard performance measure in CA simulations is the number of site updates per second: this number is roughly 8×10^4 sites per second, on a 256 FPU (8192 integer processors) CM-200. A scalability analysis (speed-up and efficiency) is difficult on such a machine as the number of processors is fixed.

A usual domain size is 256×64 , and communications between processors are negligible compared to the computation time.

The massive parallelism is very well suited to the fluid updating step as almost every cell (except the solidified ones) must be computed. As only a few lattice sites contain solid particles, the efficiency is less optimal for the snow step. This problem can be solved using a SPMD (with MPI or PVM) machine, where only the required computations are executed and the load can be explicitly balanced among the processors.

6 Results

Since no first principle theory is available for snow erosion and deposition, the only and most convincing way to assess the validity of our model is to compare its predictions with field observations which can be found in literature. Our simulations have been tested with a wide range of situations and have been shown to catch some realistic phenomena at very different spatial scales.

The fluid and the snow are modeled by virtual particles. Each virtual solid particles is not aimed at representing one real snow flake, neither one fluid particle is aimed at modeling one air molecule. The lattice automata approach relies on the complex collective behavior emerging when many particles with simple rules interact. Therefore, the correspondence between our "particle world" and real life situations is not straightforward. We shall thus proceed in four steps.

The first step is to assign a size to a lattice cell: since the number of cells is mainly limited by the available computer memory and CPU resources (a typical lattice is 512×64), the cell size is constrained by the scale of the actual problem (from 3 cm for the trench (fig. 3) or the ripples (fig. 4) up to 1.5 m for the crest deposits (fig. 2)). The second step is to get a correct Reynolds number (mainly through the calibration of C_∞ and the entry speed of the fluid in the tunnel). In the third step, setting the threshold number of deposited frozen particles before solidification of the cell gives a size to the virtual snow particle (100 part. for the crest, versus only 10 for the ripples or the trench); note that the cell size and this threshold are linked. The fourth step consists in adjusting p_{erod} , the probability of a particle ejection. The simulation is completed when the deposit reaches either a steady state (ripples) or an expected volume (trench or crest).

The situations we considered are detailed in figures 2 through 4.

Figure 2 shows the snow deposited along a mountain crest, at different positions, to compare the influence of the landscape profile on the accumulation pattern. The quality of results are rather good in the two upper cases. In the last one, we may explain the poor result observed by a bad discretization of the flat lee-slope landscape and, moreover, by an insufficient re-erosion modeling of the particles leeward the crest (where the wind is weak).

In figure 3 we see the simulation of the filling of a trench excavated in a large flat area. A qualitatively good agreement is observed between the model and reality, mainly for the first part of the experiment (growth of two deposition peaks), before the wind has slowed down in the outdoor experiment. Note that others of our numerical simulations show an accumulation on the right hand corner.

Finally, figure 4 shows small scale patterns known as ripples occurring with both sand and snow transport. Ripples are mainly due to creeping transport. The ratio we find between the height and the spacing of the oscillations (called the *wave index*) ranges around 6 (the mean ripple height is 8 sites, and the distance between two crests around 50 sites); this value agrees with the lowest index found for sand [15] in field observations, fits well wind tunnel experiments values [10] and sand ripples in water [15]. Outdoor snow ripples are more complicated since freezing and cohesion have to be taken into account; their wave index has been measured to be around 16 [1, 3]. In agreement with real observations, we also see in our simulation that ripples move horizontally. This effect is illustrated in the figure. As observed in [20], our model also shows that large ripples can be built through the merging of smaller ones traveling faster.

In conclusion, our model not only produces some quantitatively realistic deposits, it also shows, that even the simple and intuitive rules we have used, catch the basic mechanisms that occur in snow transport. It shows that the various patterns of deposition result from the emergence of a collective effect rather than from mechanisms that have yet not been identified. Creeping, saltation or suspension are no longer three different phenomena, each requiring special treatment, they are all captured by the same erosion/transport mechanisms. Thus.

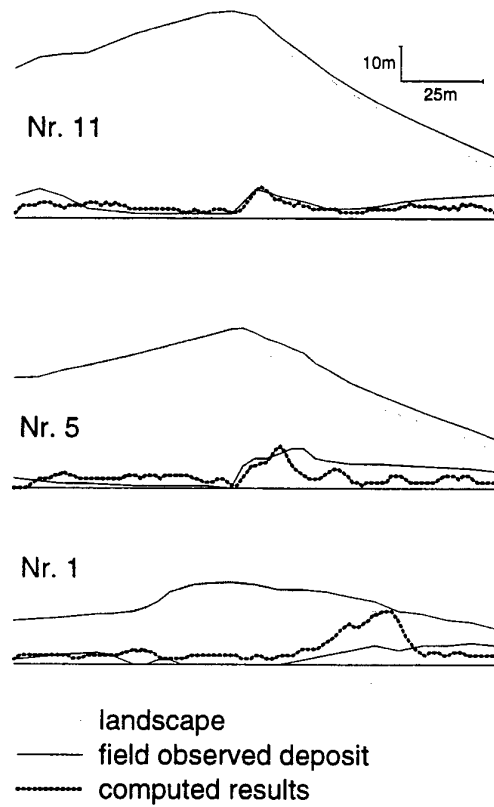


Fig. 2. Snow deposit over a mountain crest (Schwarzhorngrat near Davos), where the wind blows from left to right (at different positions along the crest). The upper part of each figure shows the ground profile and the field deposit (scale in meters) by [4] (the crests are labeled from the reference). The lower part shows only the field and the modeled deposits, stretched with a factor 2 in the z-direction. The same set of parameters has been used for the three experiments.

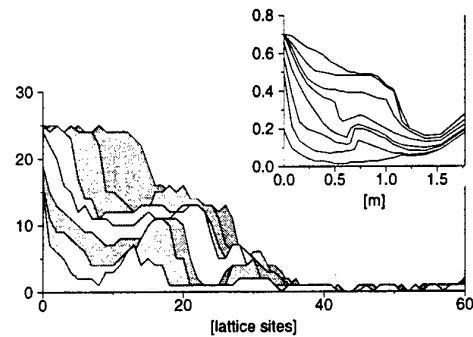


Fig. 3. A trench (0.7x1.7m, lattice spacing 0.03m) is getting buried under snowdrift. Field observations have been achieved by Kobayashi [9] (results are shown in the figure inset).

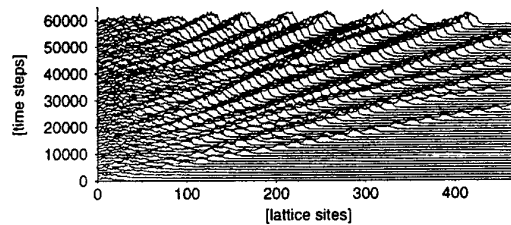


Fig. 4. Formation of ripples, as obtained from our model. Particles are continuously injected in the lower left corner of the simulation and the ripples grow spontaneously. The deposition profile is given every 1000 time steps, which makes the horizontal ripple motion quite clear (as well as the higher speed of the smaller ripples "escaping" rightwards).

our model results in a unified view of the basic laws governing the formation of snow deposition patterns.

Note that the same approach can be extended to simulated sand dune formation or sedimentation problems. To model dry sand deposits, one could neglect cohesion and allow a particle to deposit only if it occupies a stable position with respect to the solid sites underneath.

References

1. T. Castelle. *Transport de la neige par le vent en montagne: approche expérimentale du site du col du Lac Blanc*. PhD thesis, EPF Lausanne, Switzerland, 1995.
2. B. Chopard and M. Droz. *Cellular automata modeling of physical systems*. Cambridge University Press, 1998.
3. V. Cornish. *Waves of sand and snow*. T. Fisher Unwin, London, 1914.
4. P. Föhn and R. Meister. Distribution of snow drifts on ridge slopes: measurements and theoretical approximations. *Annals of Glaciology*, 4:52-57, 1983.
5. U. Frish, B. Hasslacher., and Y. Pommeau. Lattice-gas automata for the Navier-Stokes equation. *Phys. Rev. Lett.*, 56:1505, 1986.
6. W. D. Hillis. Richard feynman and the connection machine. *Physics Today*, 42(2):78, February 1989.
7. S. Hou, J. Sterling, S. Chen, and G. Doolen. A lattice subgrid model for high Reynolds number flows. *Fields Institute Communications*, 6:151-166, 1996.
8. L. Kadanoff. On two levels. *Physics Today*, 39(9):7-9, September 1986.
9. D. Kobayashi. Studies of snow transport in low-level drifting snow. *Contributions from the Institute of Low Temperature Science*, Series A(24):1-58, 1972.
10. H. Martinez. *Contribution à la modélisation du transport éolien de particules. Mesures de profils de concentration en soufflerie diphasique*. PhD thesis, Grenoble I, february 1996.
11. A. Masselot and B. Chopard. A lattice boltzmann model for snow transport and deposition. *Europhysics Letters*, 1998. To appear.
12. J. Pomeroy. *Wind Transport of Snow*. PhD thesis, University of Saskatchewan, Saskatoon, 1988. 226 p.
13. Y. Qian, D. d'Humieres, and P. Lallemand. Lattice BGK for Navier-Stokes equation. *Europhysics Letters*, 17(6):479-484, 1992.
14. Y. Qian, S. Succi, and S. Orszag. Recent advances in lattice Boltzmann computing. In D. Stauffer, editor, *Annual Reviews of Computational Physics III*, pages 195-242. World Scientific, 1996.
15. R. Sharp. Wind ripples. *Journal of Geology*, 71:617-636, 1963.
16. P.-A. Sundsbø and E. Hansen. Modelling and numerical simulation of snow drift around snow fences. In *ICSE-3, Sendai*, 1996.
17. R. Tabler. Self-similarity of wind profiles in blowing snow allows outdoor modelling. *Journal of glaciology*, 26(94):421-434, 1980.
18. M. Takeuchi. Vertical profile and horizontal increase of drift snow transport. *Journal of Glaciology*, 26(94):481-492, 1980.
19. D. Tritton. *Physical fluid dynamics*. Clarendon Press, 1988.
20. B. Werner and D. Gillespie. Fundamentally discrete stochastic model for wind ripple dynamics. *Physical Review Letters*, 71:3230, 1993.

We thank Dieter Issler and Peter Gauer from the Federal Institute for Snow and Avalanche Research in Davos for stimulating discussions. This research is supported by the Swiss National Science Foundation.

Some Concepts of the software package FEAST

Ch.Becker, S.Kilian, and S.Turek

Institut für Angewandte Mathematik, Universität Heidelberg, Germany

Abstract. This paper deals with the basic principles of the new FEM software package FEAST. For the FEAST software, which is mainly designed for high-performance simulations, we explain the basic principles of the underlying numerical, algorithmic and implementation concepts. Computational examples illustrate the (expected) numerical and computational efficiency of this new software package, particularly in relation to existing approaches.

1 Introduction

Current trends in the software development for Partial Differential Equations (PDE's), and here in particular for Finite Element (FEM) approaches, go clearly towards object-oriented techniques and adaptive methods in any sense. Hereby the employed data and solver structures, and especially the matrix structures, are often in contradiction to modern hardware platforms. As a result, the observed computational efficiency is far away from expected peak rates of almost 1 GFLOP/s nowadays, and the "real life" gap will even further increase. Since high performance calculations may be only reached by explicitly exploiting "caching in" and "pipelining" in combination with sequentially stored arrays (using machine-optimized linear algebra libraries like the ESSL or PERFLIB for instance), the corresponding realization seems to be "easier" for simple Finite Difference approaches. So, the question arises how to perform similar techniques for much more sophisticated Finite Element codes?

These discrepancies between complex mathematical approaches and highly structured computational demands often lead to unreasonable calculation times for "real world" problems, e.g. *Computational Fluid Dynamics* (CFD) calculations in 3D, as can be seen from recent benchmarks [6] for commercial as well as research codes. Hence, strategies for efficiency enhancement are necessary, not only from the mathematical (algorithms, discretizations) but also from the software point of view. To realize some of these necessary improvements our new Finite Element package (project name: **FEAST** – **F**inite **E**lement **A**nalysis & **S**olution **T**ools) is under development. This package is based on the following concepts:

- (recursive) "Divide and Conquer" strategies,
- hierarchical data, solver and matrix structures,
- SCARC as generalization of multigrid and domain decomposition techniques,
- frequent use of machine-optimized linear algebra routines,

- all typical Finite Element facilities included.

The result is going to be a flexible software package with special emphasis on:

- (closer to) peak performance on modern processors,
- typical multigrid behaviour w.r.t. efficiency and robustness,
- parallelization tools directly included on low level,
- open for different adaptivity concepts,
- low storage requirements,
- application to many "real life" problems possible.

Figure 1 shows the general structure of the FEAST package:

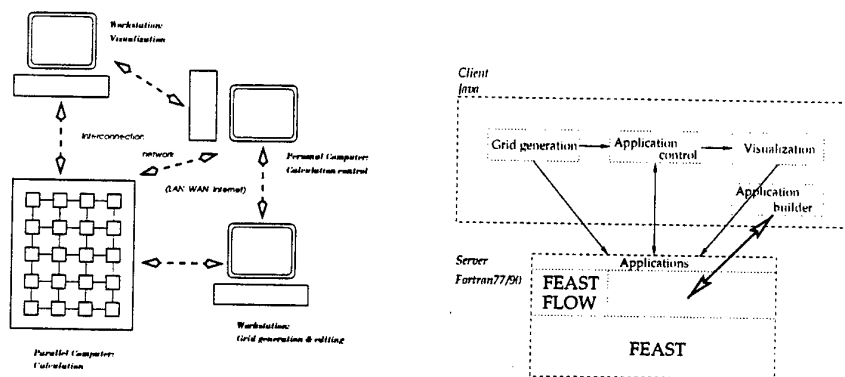


Fig. 1: FEAST structure and configuration

As programming language Fortran (77 and 90) is used. The explicit use of the two Fortran dialects arises from following observations. For Fortran77 very efficient and well-tried compilers are available which allow to exploit much of the machine performance. Further it is possible to reuse many reliable parts of the predecessor packages FEAT2D, FEAT3D and FEATFLOW [8]. On the other hand Fortran77 is not more than a better "macro assembler", the very limited language constructs make the project work very hard. Further F77 is no longer the actual standard, so what is about the support in the future? And which developer can be motivated to program in F77?

F90 on the other hand is the new standard and provides new helpful features like records, dynamic memory allocation, etc. But there are several disadvantages. The language is very overloaded and the realization of some features like pointers is not succeeded. The weighty point of criticism is the nowadays very inefficient code generation of the F90 compilers. Programs compiled with F77 and F90 show a difference in runtime up to a factor of 8, depending on algorithm and compiler.

The compromise is to implement the time critical routines from the numerical linear algebra in F77, while the administrative routines like iteration control is

based on F90. If the F90 compilers achieve the same code quality as their F77 pendants it is no problem to switch completely to F90.

The pre- and postprocessing is mainly handled by Java based program parts. Configuring a high performance computer as a FEAST server, the user shall be able to perform the remote calculation by a FEAST client. Further certain AVS/Express modules for which we agreed with AVS to include in our software package for free are provided for visualization.

In the following we give examples for "real" computational efficiency results of typical numerical tools which help to motivate our hierarchical data, solver and matrix structures. To understand these better, we illustrate shortly the corresponding solution technique ScaRC (**S**calable **R**ecursive **C**lustering) in combination with the overall "Divide and Conquer" philosophy which is essential for FEAST. We discuss how typical multigrid rates can be achieved on parallel as well as sequential computers with a very high computational efficiency.

2 Main Principles in FEAST

2.1 Hierarchical data, solver and matrix structures

One of the most important principles in FEAST is to apply consequently a (*Recursive*) *Divide and Conquer* strategy. The solution of the complete "global" problem is recursively split into smaller "independent" subproblems on "patches" as part of the complete set of unknowns. Thus the two major aims in this splitting procedure which can be performed by hand or via self-adaptive strategies are:

- *Find locally structured parts.*
- *Find locally anisotropic parts.*

Based on "small" structured subdomains on the lowest level (in fact, even one single or a small number of elements is allowed), the "higher-level" substructures are generated via clustering of "lower-level" parts such that algebraic or geometric irregularities are hidden inside the new "higher-level" patch. More background for this strategy is given in the following sections which describe the corresponding solvers related to each stage.

Figures 2 and 3 illustrate exemplarily the employed data structure for a (coarse) triangulation of a given domain and its recursive partitioning into several kinds of substructures.

According to this decomposition, a corresponding data tree – the skeleton of the partitioning strategy – describes the hierarchical decomposition process. It consists of a specific collection of elements, macros (Mxxx), matrix blocks (MB), parallel blocks (PB), subdomain blocks (SB), etc.

The *atomic units* in our decomposition are the "macros" which may be of type **structured** (as $n \times n$ collection of quadrilaterals (in 2D) with local Finite Difference data structures) or **unstructured** (any collection of elements, for instance in the case of fully adaptive local grid refinement). These "macros" (one or several) can be clustered to build a "matrix block" which contains the "local

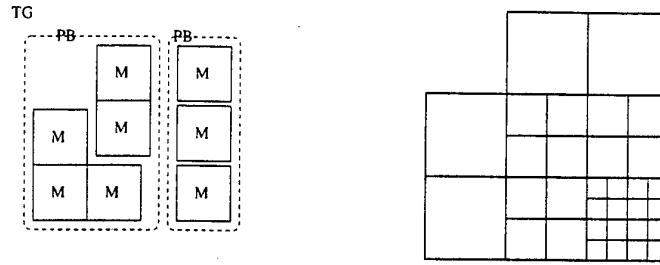


Fig. 2: FEAST domain structure

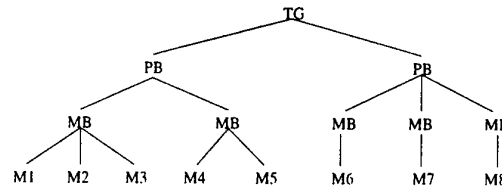


Fig. 3: FEAST data tree

matrix parts": only here is the complete matrix information stored. Higher-level constructs are "parallel blocks" (for the parallel distribution and the realization of the load balancing) and "subdomain blocks" (with special conformity rules with respect to grid refinement and applied discretization spaces). They all together build the complete domain, resp. the complete set of unknowns. It is important to realize that each stage in this hierarchical tree can act as independent "father" in relation to its "child" substructures while it is a "child" at the same time in another phase of the solution process (inside of the SCARC solver, see later).

2.2 Generalized solver strategy SCARC

In short form our long-time experience with the numerical and computational runtime behaviour of typical multigrid (MG) and Domain Decomposition (DD) solvers can be concluded as follows:

Some observations from standard multigrid approaches: While in fact the numerical convergence behaviour of (optimized) multigrid is very satisfying with respect to robustness and efficiency requirements, there still remain some "open" problems: often the parallelization of powerful recursive smoothers (like SOR or ILU) leads to performance degradations since they can be realized only in a "blockwise" sense. Thus it is often not clear how the nice numerical behaviour in sequential codes for complicated geometric structures or local anisotropies can be reached in parallel computations. And additionally, the communication overhead especially on coarser grid levels dominates the total CPU time. Even more

important is the "computational observation" that the realized performance on modern platforms is often far beyond (sometimes less than 1 %) the expected peak performance. Many codes often reach much less than 10 MFLOP/s, and this on computers which are said (by the vendors) to run with up to 1 GFLOP/s peak. The reason is simply that the single components in multigrid (smoother, defect calculation, grid transfer) perform too few arithmetic work with respect to each data exchange such that the facilities of modern superscalar architectures are poorly exploitable. In contrast, we will show that in fact 30 - 70 % can be realistic with appropriate techniques.

Some observations from standard Domain Decomposition approaches:

In contrast to standard multigrid, the parallel efficiency is much higher, at least as long as no large overlap region between processors must be exchanged. While *overlapping* DD methods do not require additional coarse grid problems (however the implementation in 3D for complicated domains or for complex Finite Element spaces is a hard job), *non-overlapping* DD approaches require certain coarse grid problems, as the BPS preconditioner for instance which may lead again to several numerical and computational problems, depending on the geometrical structure or the used discretization spaces. However the most important difference between Domain Decomposition and multigrid are the (often) much worse convergence rates of DD although at the same time more arithmetic work is done on each processor.

As a conclusion improvements are enforced by the facts that the **convergence behaviour** is often quite sensitive with respect to (local) geometric/algebraic **anisotropies** (in "real life" configurations), and that the performed **arithmetic work** (which allows the high-performance) is often restricted by (un)necessary **data exchanges**.

An additional observation which is strongly related to the previous data structure in combination with the specific hierarchical SCARC solver is illustrated in the following figure. We show the resulting "optimal" mesh from a numerical simulation of R.Becker/R.Rannacher for "Flow around the cylinder" which was adaptively refined via rigorous a-posteriori error control mechanisms specified for the required drag coefficient ([5]).



Fig. 4: "Optimal grid" via a-posteriori error estimation

As can be seen the adaptive grid refinement techniques are needed only locally, near the boundaries, while mostly regular substructures (up to 90 %) can be used in the interior of the domain. This is a quite typical result and shows that even for (more or less) complex flow simulations (here as a prototypical

example) locally blockwise "Finite Difference" techniques can be applied: these regions can be detected and exploited by the given hierarchical strategies.

The ScaRC approach consists of a separated multigrid approach for every hierarchical layer, whereby the multigrid scheme on the outest layer (subdomain layer) gives the final result. The smoothing step of the multigrid method is based on the following notation:

Smoothing on level h for $A_h x = b_h$:

- **global** outer block Jacobi scheme (with averaging operator 'M')

$$x^{l+1} = x^l - \omega_g \tilde{A}_{h,M}^{-1} (A_h x^l - b_h)$$

$$\text{with } \tilde{A}_{h,M}^{-1} := M \circ \tilde{A}_h^{-1}, \quad \tilde{A}_h^{-1} := \sum_{i=1}^N \tilde{A}_{h,i}^{-1}, \quad \tilde{A}_{h,i} := "A_h|_{\Omega_i}"$$

- "solve" **local** problems $\tilde{A}_{h,i} y_i = def_i^l := (A_h x^l - b_h)|_{\Omega_i}$ via

$$y_i^{k+1} = y_i^k - \omega_l C_{h,i}^{-1} (\tilde{A}_{h,i} y_i^k - def_i^l)$$

with $C_{h,i}^{-1}$ preconditioner for $\tilde{A}_{h,i}$, or direct!

The local smoothing operators can be a further multigrid scheme or any other scheme like Jacobi, Gauß-Seidel, ADI or ILU. The choice of the method depends on the local structure and "hardness" of the given domain. In a first step this decision is taken by the user to choose explicitly the method but in future it is planned to create an "expert system" which makes this decision widely automatically.

There are several reasons why we explicitly use **this basic iteration**:

1. This general form allows the splitting into matrix-vector multiplication, preconditioning and linear combination. All 3 components can be separately performed with high performance tools if available.
2. The explicit use of the complete defect $A_h x^l - b_h$ is advantageous for certain techniques for implementing boundary conditions (see [7]).
3. All components in standard multigrid, i.e., smoothing, defect calculation, step-length control, grid transfer, are included in this *basic iteration*.

Finally it should be explained what the notation **ScaRC** stands for:

- **Scalable**, w.r.t. the number of global ('l') and local solution steps ('k'),
- **Recursive**, since it may be applied to more than 2 global/local levels,
- **Clustering**, since fixed or adaptive blocking of substructures is possible.

For more information about ScaRC see [3], [4].

Numerical tests For examining the convergence behaviour of **ScaRC** w.r.t local anisotropies, we defined two types of anisotropic $M \times M$ -topologies, $M = 4, 8$, namely $T4(a, b, c)$ and $T8(a, b, c)$. Starting from the equidistant 4×4 -topology for the unit square, the "amount of anisotropy" can be parametrized by shifting the inner x- coordinates a, b and c to the left side of the domain as shown in figure 5, whereas the y-coordinates retain their old positions of 0.25, 0.5 and 0.75. The corresponding 8×8 -topology is obtained by one regular refinement of the 4×4 -topology.

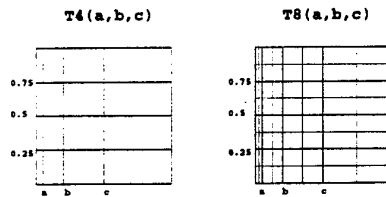


Fig. 5: Parametrizable anisotropic 4×4 - and 8×8 -topologies

Further, we added a local refinement procedure which provides an additional local anisotropic refinement of the single macros corresponding to some given input parameters (r_1, r_2, r_3) , which indicate the amount of local distortion. This procedure is designed in a special way, such that it allows the use of the standard multigrid transfer operators.

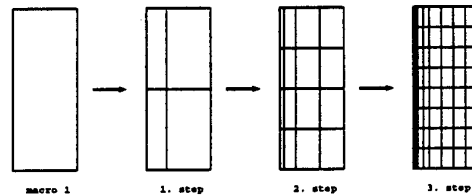


Fig. 6: Anisotropic refinement of a single macro

Figure 6 shows the refinement of the left macros in figure 5, where the single elements are shifted to the left. By means of these both procedures, arbitrarily small step sizes h_{min} , i.e., large aspect ratios AR are obtained.

For different smoothing techniques (global Jacobi, blockwise SOR, **ScaRC**) and various parameter settings of (a, b, c) , Table 1 shows the required number of multigrid iterations $\#$ for a relative accuracy of 10^{-6} and corresponding convergence rates ρ . The term N_g denotes the global number of elements in one space direction, distributed equally over the single macros. For all smoothers we performed $l = 1, 2, 4$ (global) smoothing steps and 999 multigrid iterations in

(a, b, c)	N_g	l	4×4						8×8	
			<i>Jacobi</i>		<i>SOR</i>		<i>ScaRC</i>		<i>ScaRC</i>	
			#	ρ	#	ρ	#	ρ	#	ρ
$(0.25, 0.5, 0.75)$ $AR = 1$ $h_{min} \approx 3.9 \cdot 3$	128	1	12	0.301	6	0.079	6	0.076	5	0.049
		2	7	0.126	5	0.036	3	0.006	3	0.003
		4	5	0.056	4	0.017	2	0.001	2	0.001
	256	1	12	0.304	6	0.079	6	0.083	5	0.060
		2	7	0.128	5	0.036	3	0.009	3	0.005
		4	5	0.057	4	0.017	2	0.001	2	0.001
$(0.05, 0.2, 0.5)$ $AR = 5$ $h_{min} \approx 7.8 \cdot 4$	128	1	114	0.886	35	0.674	7	0.109	9	0.208
		2	59	0.790	19	0.475	4	0.018	5	0.050
		4	31	0.635	10	0.242	3	0.002	3	0.004
	256	1	121	0.892	37	0.687	7	0.112	9	0.203
		2	63	0.801	20	0.493	4	0.022	5	0.049
		4	32	0.649	11	0.266	3	0.001	3	0.004
$(0.001, 0.01, 0.1)$ $AR = 250$ $h_{min} \approx 1.6 \cdot 5$	128	1	999	0.990	999	0.986	28	0.605	79	0.839
		2	999	0.989	614	0.977	15	0.391	43	0.723
		4	999	0.987	355	0.961	8	0.160	22	0.532
	256	1	999	0.990	999	0.988	38	0.692	113	0.884
		2	999	0.990	999	0.987	21	0.516	64	0.804
		4	999	0.989	838	0.984	11	0.278	34	0.663

Table 1: Equidistant refinement on different anisotropic $M \times M$ - topologies

the maximum. In case of **ScaRC** the local problems have been solved "exactly" with pcg-methods.

Obviously the global Jacobi smoothing provides good results for the equidistant 4×4 -topology. But with increasing anisotropy the convergence rates deteriorate quite drastically. The behaviour of the blockwise SOR-smoothing is somewhat better, but tends worse as well. Only **ScaRC** seems to be able to work well for anisotropic macro structures. But due to the underlying block Jacobi character, its convergence behaviour must depend on the structure of the macro decomposition and on the number of macros.

For the settings $(a, b, c) = (0.05, 0.2, 0.5)$ and $(a, b, c) = (0.001, 0.01, 0.1)$, table 2 compares the convergence results for different choices of the local refinement parameters (r_1, r_2, r_3) . The considered cases namely $(0.5, 0.5, 0.5)/(0.25, 0.25, 0.5)$ (which have to be compared with the equidistant case $(0.5, 1.0, 1.0)$ from the previous table) involve a moderately to strongly anisotropic refinement of the most left hand side macros of the topology, up to a finest mesh size of $h_{min} \approx 4 \cdot 10^{-9}$.

Here, the global Jacobi smoother leads to divergence in case of strongly anisotropic refinement, and the blockwise SOR smoothing produces relatively bad convergence rates, as well. In contrast, **ScaRC** provides the same rates for all kind of local anisotropies, i.e., does not deteriorate with increasing local anisotropy (in contrast to the global macro structure).

(r_1, r_2, r_3)	N_g	l	4x4						8x8	
			Jacobi		SOR		ScaRC		ScaRC	
			#	ρ	#	ρ	#	ρ	#	ρ
(0.25, 0.25, 0.5)	128	1	\nearrow	\nearrow	80	0.841	7	0.108	7	0.124
		2	134	0.902	43	0.725	4	0.018	4	0.020
		4	70	0.820	24	0.558	3	0.001	3	0.004
	256	1	\nearrow	\nearrow	167	0.921	7	0.112	7	0.121
		2	284	0.952	92	0.860	4	0.021	4	0.020
		4	151	0.912	51	0.762	3	0.001	3	0.003
(0.25, 0.25, 0.5)	128	1	\nearrow	\nearrow	470	0.971	24	0.557	62	0.800
		2	\nearrow	\nearrow	257	0.948	13	0.338	33	0.658
		4	430	0.968	144	0.908	7	0.112	18	0.451
	256	1	\nearrow	\nearrow	955	0.986	32	0.649	87	0.853
		2	\nearrow	\nearrow	539	0.975	18	0.463	49	0.752
		4	947	0.986	313	0.957	10	0.235	26	0.582

Table 2: Anisotropic refinement for $(a, b, c) = (0.05, 0.2, 0.5)$ and $(a, b, c) = (0.001, 0.01, 0.1)$

2.3 High Performance Linear Algebra

One of the main ideas behind the described (*Recursive*) *Divide and Conquer* approach in combination with the SCARC solver technology is to detect "locally structured parts". In these "local subdomains" we apply consequently "highly structured tools" as typical for Finite Difference approaches: line- or rowwise numbering of unknowns and storing of matrices as sparse bands (however the matrix entries are calculated via the Finite Element modules). As a result we have "optimal" data structures on each of these patches (which often correspond to the former introduced "matrix blocks") and we can perform very powerful linear algebra tools which explicitly exploit the high performance of specific machine-optimized libraries (i.e. ESSL, PERFLIB).

We have performed several tests for different tasks and techniques in numerical linear algebra on some selected hardware platforms. In all cases we attempted to use "optimal" compiler options and machine-optimized linear algebra libraries like the ESSL or PERFLIB. Only in the case of the Pentium II we had to perform the Gaussian Elimination with the Fortran sources exclusively which might explain the worse rates.

Gaussian Elimination: While Gaussian Elimination (GE) is presented only to demonstrate the (potentially) available performance of the given processors (often several hundreds of MFLOP/s which are really measured), we are much more interested in the realistic runtime behaviour of several matrix-vector multiplication (MV) techniques. The measured MFLOP for the Gaussian Elimination are for a dense matrix (analogously to the standard *linpack* test).

Matrix-vector multiplication: We examine more carefully the following variants which all are typical in the context of iterative schemes with sparse matrices. The test matrix is a typical 9-point stencil ("discretized Poisson operator"). We perform tests for two different vector lengths N and give the measured MFLOP rates which are all calculated via $20 \times N/\text{time}$ (for MV), resp., $2 \times N/\text{time}$ (for DAXPY (linear combination)).

Sparse MV: SMV

The *sparse MV* technique is the standard technique in Finite Element codes (and others), also well known as "compact storage" technique or similar: the matrix plus index arrays or lists are stored as long arrays containing the nonzero elements only. While this approach can be applied for arbitrary meshes and numberings of the unknowns, no explicit advantage of the linewise numbering can be exploited. We expect a massive loss of performance with respect to the possible peak rates since — at least for larger problems — no "caching in" and "pipelining" can be exploited such that the higher cost of memory access will dominate the resulting MFLOP rates.

Banded MV: BMV

A "natural" way to improve the *sparse MV* is to exploit that the matrix is a banded matrix with 9 bands only. Hence the matrix-vector multiplication is rewritten such that now "band after band" are applied. The obvious advantage of this *banded MV* approach is that these tasks can be performed on the basis of BLAS1-like routines which may exploit the vectorization facilities of many processors (particularly on vector computers). However for "long" vector lengths the improvements can be absolutely disappointing: For the recent workstation/PC chip technology the processor cache dominates the resulting efficiency!

Banded blocked MV: BBMVA, BBMVL, BBMVC

The final step towards highly efficient components is to rearrange the matrix-vector multiplication in a "blockwise" sense: for a certain set of unknowns, a corresponding part of the matrix is treated such that cache-optimized and fully vectorized operations can be performed. This procedure is called "BLAS 2+"-style since in fact certain techniques for dense matrices which are based on routines from the BLAS2, resp., BLAS3 library, have now been developed for such sparse banded matrices. The exact procedure has to be carefully developed in dependence of the underlying FEM discretization, and a more detailed description can be found in [2].

While BBMVA has to be applied in the case of arbitrary matrix coefficients, BBMVL and BBMVC are modified versions which can be used under certain circumstances only (see [2] for technical details). For example PDE's with constant coefficients as the Poisson operator but on a mesh which is adapted in one special direction only, allow the use of BBMVL: This is often the case for the Pressure-Poisson problem in flow simulations (see [7]) on boundary adapted meshes. Additionally version BBMVC may be applied for PDE's with constant coefficients on meshes with equidistant mesh distribution in each (local) direction separately: This is typical for tensor product meshes in the interior domain where the solution is mostly smooth.

Computational results The following table illustrates the above discussed linear algebra routines with their performance rates. For further results see [1].

Computer	N	$\frac{2N}{T}$	$\frac{20N}{T}$					GE
		DAXPY	SMV	BMV	BBMVA	BBMVL	BBMVC	
IBM RS6000 (166 Mhz) 'SP2'	4K	420	54	154	154	196	240	365
	64K	100	49	71	140	206	234	374
	256K	100	50	71	143	189	240	374
DEC Alpha (433 Mhz) 'CRAY T3E'	4K	390	34	46	81	158	223	309
	64K	60	22	42	63	117	189	409
	256K	60	22	42	67	110	196	454
SUN U450 (250 Mhz)	4K	165	35	102	100	99	127	246
	64K	123	15	21	35	85	119	249
	256K	30	14	17	36	72	99	282
INTEL PII (233 Mhz) 'Home PC'	4K	43	20	28	24	56	60	34
	64K	21	11	14	20	37	47	27
	256K	19	-	-	-	-	-	-

Table 3: Benchmark results

2.4 Several adaptivity concepts

As typical for modern FEM packages, we directly incorporate certain tools for grid generation which allow an easy handling of local and global refinement or coarsening strategies: **adaptive mesh moving**, **macro adaptivity** and **fully local adaptivity**.

Adaptive strategies for moving mesh points, along boundaries or inner structures, allow the same logic structure in each "macro block", and hence the shown performance rates can be preserved. Additionally, we work with adaptivity concepts related to each "macro block". Allowing "blind" or "slave macro nodes" preserves the high-performance facilities in each "matrix block", and is a good compromise between fully local adaptivity and optimal efficiency through structured data. Only in that case, that these concepts do not lead to satisfying results, certain macros will loose their "highly structured" features through the (local) use of fully adaptive techniques. On these (hopefully) few patches, the standard "sparse" techniques for unstructured meshes have to be applied.

2.5 Direct integration of parallelism

Most software packages are designed for sequential algorithms to solve a given PDE problem, and the subsequent parallelization of certain methods takes often unproportionately long. In fact it is easy to say, but hard to realize with most software packages. However the more important step, which makes parallelization much more easier, is the design of the SCARC solver according to the hierarchical decomposition in different stages. Indeed from an algorithmic

point of view, our sequential and parallel versions differ only as analogously Jacobi- and Gauß-Seidel-like schemes work differently. Hence all parallel executions can be identically simulated on single processors which however can additionally improve their numerical behaviour with respect to efficiency and robustness through Gauß-Seidel-like mechanisms.

Hence we only provide in FEAST the "software" tools for including parallelism on low level, while the "numerical parallelism" is incorporated via our SCARC solver and the hierarchical "tree structure". However what will be "non-standard" is our concept of (adaptive) parallel loadbalancing which is oriented in "total numerical efficiency" (that means, "how much processor work is spent to achieve a certain accuracy, depending on the local configuration") in contrast to the "classical" criterion of equilibrating the number of local unknowns (see [2] for detailed information and examples in FEAST).

3 Pre- and Postprocessing

3.1 General remarks

As remarked in the introduction the pre- and postprocessing should be realized in main tasks by a general framework of Java based programs called DeVISO. DeVISO means "Design & Visualization Software Resource". This framework is intended to perform the main tasks grid generation and editing, control of the calculation and visualization of the results. These main tasks use the same ground classes (called DFC - DeVISO Foundation Classes) and the same user interface, so the access to the underlying numerical core parts are performed in the same manner.

As intended in the introduction the various subtasks can be performed on several machines which communicates over a network system. This allows the user to choose the suitable system for the corresponding task, e.g. a Silicon Graphics workstation for the visualization. The access to a parallel computing system should also be performed by a Java program. This allows not only the developer of a numerical code to use a parallel computer.

DeVISO is planned to be an "open system" for the developing of pre- and postprocessing tools for FEM packages. The DeVISO foundation classes contain the basic tools to handle and administrate FEM typical structures. Further applications could realize e.g. further visualization procedures and adaptations to several parallel computer systems.

For this project Java as implementation environment is been choosen. Though Java is a relative "young" programming language the advantages of this system are significant. The "write once, run anywhere" capability reduces the implementation effort widely against combinations like C/C++/OpenGL. It exist only one program which runs without any modification on several different configurations like Unix workstations, Linux PCs, Windows PCs, Macintoshs and many more. A further advantage is the core class library for various subareas like file handling, network functions, visualization and user interface facilities. These classes

are easy to use and produce an pleasing output. The use of additional tools like applications builders is not necessary. The most disadvantage of nowadays Java implementations is the relative low performance because of the fact that Java is an interpreted language. However further developments like more sophisticated interpreter with Just-In-Time compiling facilities and especially the native Java processor will hopefully close this performance leak.

3.2 Preprocessing: DeVISOGrid

This subprogram should support the generation and editing of 2D domains. The two main parts are the description of the domain boundary and the generation of the grid structure. The program supports several boundary elements like lines, arcs and splines, further it is planned to add a segment which consists of an Fortran subroutine which describes a parametrization. This allows to use an analytic description. Several triangular and quadrilateral elements are supported. Extensive editing possibilities allow the user to delete, move and adjust the boundaries and elements. For the future it is planned to implement simply automatic grid generators for producing coarse grids. As further tasks this program should be able to read many formats from other tools like CAD systems and professional grid generators and prepare this data for the use in the calculation process.

3.3 Processing: DeVISOControl

DeVISOControl enables the user to control the calculation und to follow the calculation progress. Main tasks of this program part are the distribution of the macros to the processing nodes (at the moment manually, in future automatically), the collecting and displaying of the log information of the processing nodes and finally the configuration of the ScaRC algorithm with respect to the selection of smoothing/preconditioning methods on a given hierarchical layer, the size of smoothing steps and the exit criterion. Furthermore this part builds the interface to the other DeVISO parts for the pre- and postprocessing. From the control part the grid program is invoked, a grid is editing. For this grid the user selects the desired solution method and visualize finally the results with the DeVISOVision program.

3.4 Postprocessing: DeVISOVision

The last part in the current project is the DeVISOVision program which performs the visualization task. The program offers several techniques to visualize the results of the calculation like shading techniques, isolines, particle tracing (planned). Furthermore it contains an animation module to create animations for nonstationary problems. The result of the animation can be stored in several formats like MPEG and AnimatedGIF.

4 Conclusions and Outlook

We expect the first version of FEAST for end of 1998, but most of the "numerical" and "computational" ingredients have already been successfully realized in several test implementations (see the references). The actual status of the FEAST project and further information can always be obtained from our web page:

<http://gaia.iwr.uni-heidelberg.de/~featflow>

Nevertheless, help is always welcome: for instance in implementing and testing many auxiliary components, pre- and postprocessing, "unit square" experts and "computers for performance measurements", etc.

References

1. Altieri, M., Becker, Ch., Turek, S.: On the realistic performance of components in iterative solvers, Proc. FORTWIHR Conference, Munich, March 1998, LNCSE, Springer-Verlag, to appear.
2. Becker, Ch.: The realization of Finite Element software for high-performance applications, Thesis, to appear.
3. Kilian, S.: Efficient parallel iterative solvers of SCARC-type and their application to the incompressible Navier-Stokes equations, Thesis, 1998.
4. Kilian, S., Turek, S.: An example for parallel SCARC and its application to the incompressible Navier-Stokes equations, Proc. ENUMATH-97, Heidelberg, October 1997.
5. Rannacher, R., Becker, R.: A Feed-Back Approach to Error Control in Finite Element Methods: Basic Analysis and Examples, Preprint 96-52, University of Heidelberg, SFB 359, 1996.
6. Schäfer, M., Rannacher, R., Turek, S.: Evaluation of a CFD Benchmark for Laminar Flows, Proc. ENUMATH-97, Heidelberg, October 1997.
7. Turek, S.: Efficient solvers for incompressible flow problems: An algorithmic approach in view of computational aspects, LNCSE 2, Springer-Verlag, 1998.
8. Turek, S.: FEATFLOW . Finite element software for the incompressible Navier-Stokes equations: User Manual, Release 1.1, 1998

Dynamic Routing Balancing in Parallel Computer Interconnection Networks¹

D.Franco, I.Garcés, E.Luque

Unitat d'Arquitectura d'Ordinadors i Sistemes Operatius-Departament d'Informàtica
Universitat Autònoma de Barcelona - 08193-Bellaterra, Barcelona, Spain
Ph. +34-3-581.19.90 Fx.+34-3-581.24.78
E-mail: {iarq23,d.franco,iinfid}@cc.uab.es; Web: <http://aows1.uab.es>

Abstract. In creating interconnection networks, an efficient design is crucial because of its impact on the parallel computer performance. A routing scheme that minimises contention and avoids the formation of hot-spots should be included in the design. Static schemes are not able to adapt to traffic conditions. We have developed a new method to uniformly distribute traffic over the network called Distributed Routing Balancing (DRB) that is based on limited and load-controlled path expansion in order to maintain a low message latency. The method uniformly balances the communication load between all links of the interconnection network and maintains a controlled latency, provided that total bandwidth requirements do not exceed the total link bandwidth available in the interconnection network. DRB defines how to create alternative paths to expand single paths (expanded path definition) and when to use them depending on traffic load (expanded path selection policies). We explain the DRB principles and show the performance evaluation of the method carried out by simulation.

1. Introduction

In the evolution of multi-computers, communication performance becomes more and more important. One of the most crucial problems that affects performance in communications is message contention. A sustained contention can produce hotspots [Pfi85]. A hotspot is a saturated region of the network, i.e. there exists more bandwidth demand than the network can offer and then, messages that enter this region suffer a very high latency, while other regions of the network can be less loaded, or even far away from saturation. The problem here is that there exists a poor communication load distribution and that, although the total communication bandwidth requirements do not surpass the total offered bandwidth of the interconnection network, this uneven distribution causes saturated points as if the whole interconnection network were collapsed. In addition, the hot-spot propagates rapidly to contiguous areas in a domino effect, which is even worse in the case of

¹ This work has been supported by the Spanish Comisión Interministerial de Ciencia y Tecnología (CICYT) under contract number TIC 95/0868.

wormhole routing because a blocked packet occupies a large number of links spread in the network.

Latency must be avoided in order to make communications faster, but some amount of latency can be tolerated and it is much more important to avoid big latency variations. This is because latency can be hidden by the mapping task assigning an excess of parallelism, i.e. having enough processes per processor and scheduling any ready process while other processes wait for their messages. However, in order to be able to assign processes to processors correctly, the mapping task must know the process computation and communication volumes and, to some extent, the latency that messages will suffer. But if latency undergoes big unpredictable variations from the expected values, due to hotspots, for example, the mapping will fail and idle processors will appear, increasing the total execution time of the application. This is the reason for the importance of a low and uniform contention latency. In addition, in Distributed Shared Memory Multicomputers latency uniformity is a key issue for scheduling and mapping in such systems.

In order to avoid hotspot generation, static or oblivious routing can not provide any help because, under this routing, a message route is completely determined by the source-destination pair, independent of traffic conditions. Therefore, other mechanisms have been developed to avoid hotspot generation in interconnection networks like adaptive routing algorithms that try to adapt to traffic conditions such as Planar Adaptive Routing [CK92], the Turn Model [NG92], Duato's Algorithm [Dua93], Compressionless Routing [KLC94], Chaos Routing [Ksn91], Random Routing [Val81] [May93] and other methods presented in [DYN97]. The main disadvantage of adaptive routing is the high overhead because of information monitoring, path changing and the necessity to guarantee deadlock, livelock and starvation freedom. These drawbacks have limited the implementation of these techniques in commercial machines.

The work presented in this paper focuses on developing new methods to distribute paths in the interconnection network using network-load controlled path expansion. The method is called Distributed Routing Balancing (DRB) and its objective is to uniformly balance traffic load over all paths in the whole interconnection network by creating alternative paths between each source and the destination nodes in order to maintain a low message latency. DRB defines how to create alternative paths to expand single paths (multi-lane path definition) and when to use them depending on traffic load (multi-lane path selection policy).

The next section explains the Distributed Routing Balancing technique. DRB has two components: first, a systematic methodology to generate the multi-lane paths and second, policies to monitor traffic load and select multi-lane paths to get the message distribution according to traffic load. Both are explained in Section 3 and Section 4, respectively. Sections 5 presents the evaluation of the first DRB component and Section 6 the validation of the selection policies. Section 7 presents the conclusions.

2. Distributed Routing Balancing

Distributed Routing Balancing is a method to create alternative source-destination paths in the interconnection network using a load-controlled path expansion. DRB distributes every source-destination message load over a multi-lane path made of several paths. The objective of DRB is a uniform distribution of the traffic load over the whole interconnection network in order to maintain a low message latency and avoid the generation of hotspots. When a single source-destination path is becoming saturated, the method looks for low loaded paths to form a multi-lane path. This distribution will maintain a uniform and low latency on the whole interconnection network provided that total communication bandwidth demand does not exceed interconnection network capacity.

The DRB method fulfils the following objectives:

- 1 Reduction of the message latency under a certain threshold value by varying the number of alternative paths used by the source-destination pair, while maintaining a uniform latency for all messages.
- 2 Minimisation of path-lengthening. This is important for Store&Forward networks because Transmission Delay depends directly on the message path lengths. For Wormhole and Cut-Through flow controls, it is important because the more nodes used by the message, the more collisions with other messages, causing latency increments and more bandwidth use.
- 3 Maximisation of the use of the source and destination node links (node grade), distributing messages fairly over all processor links.

In order to show how DRB works to create and use alternative paths, we make the following definitions:

Definition 0:

An *interconnection network I* is defined as a directed graph $I=(N,E)$, where N is a set of nodes $N=\bigcup_{i=0}^{MaxN} N_i$ and E a set of links. Every node is composed of a router and is connected to other nodes by means of links. In regular networks, a *regular topology* is defined, with a *dimension* and *size*. For example, for k -ary n -cubes [Da190], n is the dimension and k the size. For irregular networks, an *irregular topology* is defined.

- If two nodes N_i and N_j are directly connected by a link, then, N_i and N_j are *adjacent nodes*.
- $Distance(N_i, N_j)$ is the minimum number of links that must be traversed to go from N_i to N_j according to the graph I .

- A path $P(N_i, N_j)$ between two nodes N_i and N_j is the set of nodes selected between N_i and N_j according to the minimal static routing defined for the interconnection network. N_i is the *source* node and N_j the *destination* node. **Length** of a path P $Length(P)$ is the number of links traversed between N_i and N_j .

Definition 1:

A Supernode $S(type, size, N_0^S, V(S)) = \bigcup_{i=0}^l N_i^S$ is defined as a structured region of

the interconnection network consisting of adjacent nodes N_i^S around a “central” node N_0^S provided that N_i^S comply with a given property specified in *type* and that $distance(N_i^S, N_0^S) \leq size$. Each node has an associated weight w_i^S stored in the array $V(S)$.

$V(S) = \{0 \leq w_i^S \leq 1 \in R; i = 0..l\}$ is a linear array of weights associated with each N_i^S where w_i^S is the weight of N_i^S and $\sum_{i=0}^l w_i^S = 1$. These weights are used by the DRB selection policy (Sect. 4). As particular cases, any single node and the whole interconnection network are Supernodes. A node can belong to more than one Supernode. Section 3 presents different Supernode types.

Definition 2:

A Multi-step Path $P_s(SOrigin, N_i^{SOrigin}, N_j^{SDest}, SDest)$ is the path generated between two Supernodes $SOrigin$ and $SDest$ as $P_s = \prod (N_0^{SOrigin}, N_i^{SOrigin}, N_j^{SDest}, N_0^{SDest}) = P(N_0^{SOrigin}, N_i^{SOrigin}) \bullet P(N_i^{SOrigin}, N_j^{SDest}) \bullet P(N_j^{SDest}, N_0^{SDest})$, where \bullet means path concatenation, composed of the following steps:

Step 1: From the central node of the Supernode $Supernode_Origin, N_0^{SOrigin}$, to a node belonging to $Supernode_Origin, N_i^{SOrigin}$.

Step 2: From the $N_i^{SOrigin}$ to a node belonging to $Supernode_Destination, N_j^{SDest}$.

Step 3: From the N_j^{SDest} to the central node of the Supernode $Supernode_Destination, N_0^{SDest}$.

In the most general case, there are three steps. However, if one of the Supernodes $Supernode_Origin = \{N_0^{SOrigin}\}$ or $Supernode_Destination = \{N_0^{SDest}\}$, the number

of steps is two; and there is one step, the one which follows static routing, if $Supernode_Origin = \{ N_0^{SO} \}$ and $Supernode_Destination = \{ N_0^{SD} \}$.

Length of a multi-step path $Length(P_s)$ is defined as the sum of each individual step length following static routing. From this definition, it can be seen that Multi-step Paths between N_0^{SO} and N_0^{SD} can be of non minimal length.

Definition 3:

A **Metapath** $P^*(Supernode_Origin, Supernode_Destination)$ is the set of all multi-step paths P_i generated between the Supernodes $Supernode_Origin$ and

$Supernode_Destination$: $P^* = \bigcup_{\forall i,j} P_s (N_0^{SO}, N_i^{SO}, N_j^{SD}, N_0^{SD})$. Suppose l

is the number of nodes of $Supernode_Origin$ and k the number of nodes of $Supernode_Destination$. The number of Multi-step Paths which compose the Metapath is $s=l*k$. **Metapath Length (ML)** is the average of all the individual multi-step path lengths that compose it, $Length(P^*) = (1/s) \sum_{\forall s} length(P_s)$

and **Metapath Relative Bandwidth (MRB)** is the number of multi-step paths s .

Now, we explain how communication is managed under DRB to get path distribution. Suppose there is a parallel program described as a collection of processes and channels and a process mapping which assigns each process to a processor. Processes are executed concurrently and communicate by channels.

The routing run-time support configures a Metapath P^* for each channel by assigning a *Source Supernode* to the source node and a *Destination Supernode* to the destination node. This function is carried out by a Metapath selection policy called Metapath Issuing through Latency Evaluation (MILE) which is described in Section 4. The Source Supernode is a **Message Scattering Area (MeSA)** from the Source node. The Destination Supernode is a **Message Gathering Area (MeGA)** to the Destination node.

Then, for each message that the source process wants to send, the channel manager in the routing run-time support, selects two nodes, one N_i^{SO} from the *Source Supernode*, and the other N_j^{SD} from the *Destination Supernode* to form a *Multi-step Path* $P_s (SO_{origin}, N_i^{SO}, N_j^{SD}, SD_{Dest})$ belonging to the Metapath P^* . These node selections are made based on the weight arrays which are used as probability distributions for each node $N_i^{SuperNode}$. Then, the message travels along the selected *Multi-step Path*.

Under this scheme, the communication between source and destination can be seen as if it were using a wider multi-lane "Metapath" of potentially higher bandwidth than the original path from a source "Supernode" and to a destination "Supernode". This

multi-lane path can be likened to a highway and the MeSA and MeGA, the highway access and exit areas, respectively.

It is important to remark that, in order to achieve an effective uniform load distribution, a global action is needed. For this reason, all source-destination nodes are able to expand their paths depending on the message traffic load between them during program execution.

The mentioned DRB Metapath Selection Policy defines the Metapath *type* and *size* to determine a Metapath of specific length and bandwidth depending on traffic conditions (Section 4). The policy needs to know the length and bandwidth of a Metapath given the type and size. Therefore, a Metapath Characterisation is needed to determine the Length and Relative Bandwidth of a Metapath given its type and size. This characterisation has been carried out by experimentation and is explained in Section 5.

Comparison with existing methods

Many adaptive methods try to modify current path when a message arrives to a congested node. This is the case, for example, of Chaos routing [KS91] which uses randomisation to misroute messages when the message is blocked. DRB does not act at the individual message level, but tries to adapt communication flow between source and destination nodes to non-congested paths.

Random routing algorithms [Val81] [May93] uniformly distribute bandwidth requirements over the whole machine, independent of the traffic pattern generated by the application but at the expense of doubling the path length. A closer view shows that paths of maximum length are not lengthened but paths of length one are lengthened, in average, up to the average distance for regular networks. So, the shortest paths are extremely affected. This is due to the method being "blind", namely it does not take into account current traffic and it distributes all messages at "brute force" over the entire machine. Although DRB shares some objectives with random routing, the difference is that DRB does not only try to maintain throughput but also maintains limited individual message latency, because path lengthening can be controlled. Random routing, however, doubles in average the lengthening with the negative effect over the latency we mentioned above. It can be seen that static routing is an extreme case of DRB in which both Supernodes, source and destination, contain only the source or destination node, respectively; and that random routing is the other extreme in which the source Supernode contains all nodes of the interconnection network.

A similar but restricted, less flexible and non-adaptive solution is offered by the IBM SP2 routing algorithm, RTG, that statically selects four paths for each source-destination node which are used in a "round-robin" fashion to more uniformly utilise the network [Sni95]. The Meiko CS-2 machine also pre-establishes all source-destination paths and select four alternative paths to balance the network traffic [Bok96].

Like other adaptive methods, DRB can introduce the possibility of deadlock in which case one of the existing techniques for deadlock avoidance should be used

(such as structured buffer-pool [Mer80], virtual channels [Dal87] or virtual networks [Yan89]) depending on network characteristics (topology, flow control, etc.).

It can be seen that, by definition DRB is livelock free, because it never produces infinite path lengths, and also starvation free, because no node is prevented from injecting their messages. Also, message ordering must be preserved and it is the system's responsibility to deliver messages belonging to the same logical channel. Message prefetching can be used to hide message disordering.

3. Supernode Types

We have defined two *Supernode types* suitable for any topology and which define a broad range of Supernodes that allow a choice according to the desired trade-offs between Metapath length (ML) and Relative Bandwidth (MRB). The first one is called *Gravity Area* and the second *Subtopology*.

The parameters *type* and *size* of the Supernode determine which nodes are included in the Supernode. A given Supernode has the following characteristics: Topological shape, Number of nodes l , Grade (number of N_i node links not connected to other N_i node links, i.e. links connected outside the Supernode) and Grade usage of the central node N_0 in the first step.

Gravity Area Supernode

The first Supernode type, $S(\text{"Gravity Area"}, \text{size}, N_0^S, V(S))$, is called *Gravity Area* and defines, for a n -grade network, a n -ary tree with the root at the central node N_0^S and a deep *size*. This type maps a tree of maximum grade over the topology. This tree expands at maximum and includes all nodes which are at distance *size* or less from the root. It is suitable for regular or irregular networks. A *Gravity Area* Supernode is the set of nodes at a distance smaller or equal than the *size* of the node N_0^S . Metapaths configured using Gravity Area Supernodes fulfil the above mentioned objectives of maximising the number of paths while minimising path-lengthening and maximising node link usage since they make use of all node links.

Subtopology Supernode:

The second Supernode type is called *Subtopology* and defines the topological shape of the Supernode. It can be applied to regular networks with a structured topology, dimension and size. A Supernode $S(\text{"Subtopology"}, \text{size}, N_0^S, V(S))$ has the same full/partial topological shape as the interconnection network but its *dimension* and/or *size* is reduced. Therefore, the *Subtopology* Supernode should be considered as a kind of topological "projection" of the network topology. For example, in a k -ary n -cube a *Subtopology* Supernode is any j -ary m -cube with $j < k$ and/or $m < n$. For Midimew networks, which are a special case of wraparound torus (k -ary 2-cubes),

Subtopology Supernodes are k-ary 1-cubes, i.e. linear structures which follow specific wraparound links.

4. Metapath Selection Policy

This section describes the Metapath selection policy to select a Metapath for source destination pairs according to the current message load. We have designed and present here a dynamic policy for DRB called Metapath Issuing through Latency Evaluation (MILE), which has been designed trying to minimise overhead and to be scalable. To this end, there is no periodic information exchange and MILE is fully distributed. It has the characteristic that, under low traffic load, the monitoring activity is minimum and the paths follow minimal static routing. The monitoring activity objective is to identify the current traffic pattern.

Policy objectives are to select the supernode *size* and *type* and to distribute the load among the Multi-step Paths of the Metapath. The policy consists of three phases: Traffic Load Monitoring, Dynamic Supernode Configuration and Multi-Step Path Selection. Traffic load monitoring is carried out by the messages. Latency suffered is recorded and carried by the message itself. The message records information of the contention it suffers at each node it traverses when it is blocked by contention with other messages. When messages arrive at their destination carrying latency information, the destination node takes a decision about the Supernode configuration depending on the contention that the messages encountered on the Metapath. This Metapath Configuration is sent to the Source node by means of an acknowledge message which distributes messages following DRB specification.

The policy algorithm pseudo-code for each source destination pair is presented as follows:

```
Traffic Load Monitoring()/*Actions performed by the messages*/
Begin
  For each hop,
    Accumulate latency
  EndFor
  Deliver latency to the Destination algorithm.
End
```

```
Dynamic Supernode Configuration(threshold latency ThL)
/*Algorithm executed at destination nodes*/
/*Threshold latency is the change latency from the flat region
to the rise region defined in Sec. 2*/
Begin
  For each Multi-step Path of the Metapath
    Receive Multi-step Latency recorded by the message itself.
  EndFor
  Order Multi-step Paths according to the Latency they suffered
  Classify Multi-step Paths as saturated or non-saturated
  depending on whether their latencies exceed the ThL or not.
  Calculate Total Latency (TL) adding each Multi-step Latency.
```

```

    Calculate Total Threshold Latency (TThL) multiplying
    TL*Metapath Relative Bandwidth(MRB)
    /*Load Balancing: Distribute traffic load among the Multi-
    step Metapaths*/
    If (TL does not exceed TThL and there exist saturated Multi-
    step Paths)

        Redistribute Supernode node  $w_i^S$  weights to move load from
        saturated Multi-step Paths to non-saturated Multi-step Paths
        ElseIf (TL does not exceed TThL and there do not exist
        saturated Multi-step Paths)
            Reduce Multi-step Paths (reducing Supernode
            configurations)

            Redistribute Supernode node  $w_i^S$  weights to move load from
            disappearing Multi-step Paths to the other Multi-step Paths
            ElseIf (TL exceed TThL)
                Add new Multi-step Paths (expanding Supernode
                configurations) as non-saturated Multi-step Paths

                Redistribute Supernode node  $w_i^S$  weights to move load from
                saturated Multi-step Paths to non-saturated Multi-step Paths
            EndIf
            Send new Supernode Configurations and weights to the Source
            node by means of an acknowledge message.
    End Destination

```

```

Multi-Step Path Selection() /*Actions executed by sender nodes*/
Begin
    Receive new Supernode configurations
    Distribute subsequent messages among the Supernode nodes
     $N_i^S$ , according to their corresponding weights  $w_i^S$ .
End

```

This DRB MILE takes advantage of the spatial and temporal locality of parallel program communications, like cache memory systems do with memory references. The algorithm adapts the Metapath configurations to the current traffic pattern. While this pattern is constant, latencies will be low and the MILE is not activated. If the application changes to a new traffic pattern and message latencies change, the MILE will adapt Metapaths to the new situation. DRB is useful for persistent communication patterns which are the ones which can cause the worst hotspot situations. This Metapath adaptability is specific and can be different for each source-destination pair depending on their static distance or latency conditions.

Memory space and the execution time overhead of the policy is very low because the implied actions are very simple. In addition, these activities are executed a number of times which is linearly dependent of the number of logical channels of the application and the number of messages. Regarding the time overhead, it can be seen that monitoring is just latency record by the message itself, i.e. storing a few integers, and that the decision algorithm is a local and simple computation applied only each time a latency rise is detected. Regarding the space overhead, the latency record is

one or a few integers that the message carries itself in its header, and the new Supernode configuration information is a short message of integers.

5. Metapath characterisation

This section explains the metapath characterisation that has been carried out to determine the Metapath Length (ML) and Metapath Relative Bandwidth (MRB) of a Metapath given its *type* and *size*. A series of experiments have been carried out for all Metapath types and sizes and for k-ary n-cubes and Midimews from 8 to 64K nodes.

For a given *topology* and metapath *type* and *size*, the experiment consisted in calculating the average ML and average MRB by averaging ML and MRB for all generated Metapaths when changing the source and destination nodes.

This average ML for a specific Metapath is considered as the average network distance of the interconnection network under the routing defined by the Metapath. This average network distance has been compared to the average network distance for static and random routings.

In addition, the Standard deviation of the lengths of the Multi-step Paths which compose the Metapath was calculated. The standard deviation shows the uniform fairness of the method in relation to path lengthening.

As an example of the results obtained, fig. 1 shows a chart for a 1024-node (32x32) 2D Torus and a 1024 10D HyperCube. The chart shows the Relative Metapath Average Length, i.e. the percentage relation between the average ML and average network distance for static routing, and the Metapath Relative Bandwidth. The X axis represents different Metapaths sizes for the Metapaths $M(S_o, S_d)$ where S_o ("Gravity Area", size, $N_i^{S_o}, V(S)$) and S_d ("Gravity Area", size, $N_j^{S_d}, V(S)$). Each Xi axis point is the average of the MLs for the Metapaths of size=Xi generated for all source and destination nodes.

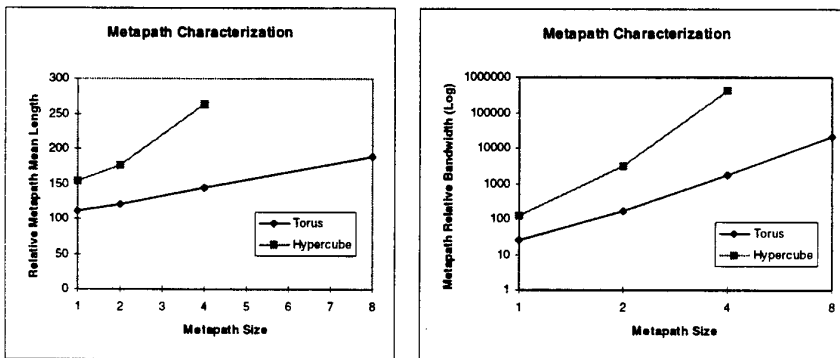


Fig. 1. Metapath characterisation

We can make the following observations from the above charts for Gravity Area methods. Looking at Relative Metapath Average Length values, it can be seen that Metapath lengthening growth is proportional to the supernode size for Torus and HyperCubes. Regarding Metapath Relative Bandwidth, it can be seen that it has a much higher growth with respect to metapath size than Metapath Length.

In addition, the results show a similar behaviour for equivalent supernodes in different topologies, which demonstrates method uniformity. Besides the broad range of alternatives offered by the methods, gravity area methods offer a higher extra bandwidth/ latency rate than Subtopology methods. Similar results have been found for all topology sizes from 9 to 64K processors which proves the good scalability of the methods. A complete presentation of these results is found in [Gar97].

6. DRB MILE Policy Evaluation

This section shows the results for different traffic patterns and network loads with a fixed message length and compare the performance with that of static routing.

The simulations consisted of sending packets through the network links according to a specific traffic pattern. The simulations were conducted for an 8x8 torus with bi-directional links. We have assumed wormhole flow control and 10 flits per packet. Each link was designed to have only one flit buffer associated with it. The packet generation rate followed an exponential distribution whose average was the message interarrival time. The results were run many times with different seeds and were observed to be consistent. The simulation was carried out for 100,000 packets. The effects of the first 20,000 delivered packets are not included in the results in order to lessen the transient effects in the simulations.

We have chosen some of the communication patterns commonly used to evaluate interconnection networks [DYN97]: Uniform, hot-spot, Matrix transposition and butterfly.

We have studied the average communication latency, the average throughput of the network, and the traffic load distribution in the network. The communication latency was measured as the total time the packets have to wait to access the link from source to destination. The throughput was calculated as the percentage relation between the accepted and the applied communication loads measured as number of messages per unit time. In order to show the traffic load distribution, we measure the average latency in each link of the network. The experiments were conducted for a range of communication traffic load from low load to saturation.

Results

Under uniform traffic, there does not exist load unbalance and, therefore, DRB routing does not modify the load distribution of the network load, resulting in almost the same average latency and average throughput of the network for all ranges of load. This is the behaviour expected according to DRB's definition which can not

improve this situation. We do not show the figures for this case. For the other three traffic patterns the behaviour changes in a very different way.

Fig.2 shows the latency results for the hot-spot traffic, the bit-reversal and butterfly traffic patterns. DRB routing demonstrates better performance. It can be seen that at low load rates, DRB behaves nearly equal to static routing. This means that the DRB method does not charge the network when it is not necessary. While load is increasing, latency improvements are increasing too, resulting in latency reductions bigger than 50% at the highest load.

At the same time as these latency improvements are achieved, the throughput is increased as can be seen in Fig. 2 for all traffic patterns. The throughput is improved up to a 50%. The conclusions are that more messages are sent and with less latency and that the network saturation point is reached at higher load rates because DRB routing maintains uniform load distribution getting a better use of network resources for all tested traffic patterns.

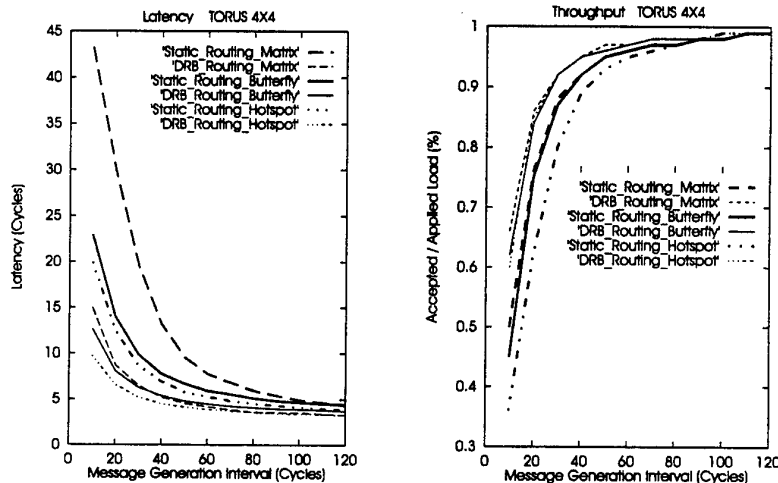


Fig. 2. Performance results for DRB routing: average latency, average throughput.

In order to show how DRB Routing distributes load and eliminates hot-spots, Fig. 3 shows the latency surface for network links for the hot-spot traffic pattern at a load rate of 30 cycles as message generation interval. Each grid point represents the average latency of the node links of the torus. It can be seen that, using Static Routing, big hot-spots appear in the network while other regions of the network are only slightly used. The maximum average latency in the hot-spots is around 15 cycles. When using DRB Routing, these hot-spots are effectively eliminated because the load excess of the hot-spot nodes is distributed among other links. The maximum average latency in this case is about 3.5 cycles.

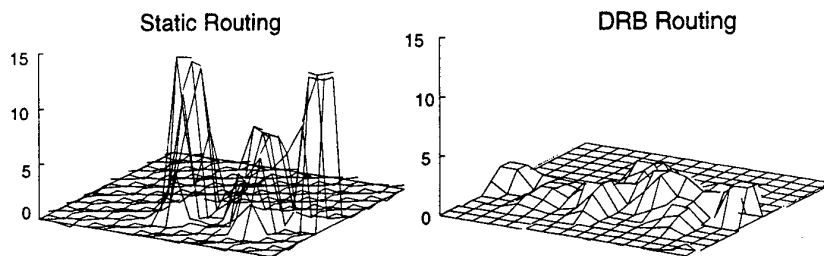


Fig. 3. Latency distribution for the hot-spot pattern

7. Conclusions

Distributed Routing Balancing is a new method for message traffic distribution in interconnection networks. DRB has been developed to try to fulfil the design objectives for parallel computer interconnection networks. These objectives are all-to-all connection and low and uniform latency between any pair of nodes and under any message traffic load. Traffic distribution is achieved by defining Supernodes which firstly send messages to an intermediate destination before sending them to their final destination. Two Supernodes are defined, the first one is centred at the source node and the second at the destination node. Only one or both kinds can be used resulting in one or two intermediate destinations for each source-destination pair.

DRB has two components. The first component is Supernode definition and the second is Metapath selection.

Supernode definition has been explained and its parameters (latency/bandwidth) characterised experimentally for its subsequent selection in the adaptive phase. The new type of Supernode Gravity Area turns out to be more interesting than that defined by topological analogy, because it maximises link usage, increasing the output width from source/destination, not only along the message path. Therefore, a methodology for Supernode definition has been created for each topology. DRB offers a set of alternative paths to choose from, depending on the trade-offs between throughput and latency.

The second component of DRB are the policies to select a specific Supernode. The dynamic policy we present monitors traffic load and dynamically configures Supernode parameters depending on the temporary requirements of message load in the network. The policy does not waste significant computation or communication resources because they are fully distributed, and the monitoring and decision overhead is linearly dependent of the number of messages in the network. DRB is useful for persistent communication patterns which are the ones which can cause the worst hotspot situations. The evaluation done to validate DRB shows us very good improvements in latency, effectively eliminating hotspots from the network.

Bibliography

- [Bei92] R. Beivide et al. "Optimal Distance Networks of Low Degree for Parallel Computers". IEEE Trans. on Computers. Vol. 40. N. 10. pp. 1109-1124. Oct 1992.
- [Bok96] Bokhari S. "Multiphase Complete Exchange on Paragon, SP2 and CS2". IEEE Parallel and Distributed Technology, Vol.4, N.3, Fall 1996, pp. 45-49
- [CK92] Chien AA, Kim JH, "Planar Adaptive Routing: Low-Cost Adaptive Networks for Multiprocessors". Proc. 19th Symp. on Computer Architecture. May 1992, pp. 268-277
- [Dal87] Dally WJ, Seitz CL. "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks" IEEE Trans. On Computers. V. C-36. N.5, May 1987. 547-553.
- [Dal90] Dally WJ. "Performance analysis of k-ary n-cube interconnection networks". IEEE Trans. On Comput. Vol. 39. Jun. 1990.
- [Dua93] Duato J. "A new theory of Dead-lock free adaptive routing in wormhole networks" IEEE Transactions on Parallel and Distributed Systems, 4(12), Dec 1993, pp.1320-1331
- [DYN97] Duato J, Yalamanchili S, Ni L. "Interconnection Networks, an Engineering Approach". IEEE Computer Society Press. 1997
- [Gar97] Garces I, Franco D, Luque E "Improving Parallel Computer Communication: Dynamic Routing Balancing". Proc. 6th Euromicro Workshop on Parallel and Distributed Processing. (IEEE-Euromicro) PDP98. Madrid. Spain. January 21-23, 1998 p. 111-119
- [KLC94] Kim J, Liu Z, Chien A. "Comprehensionless Routing: A Framework for Adaptive and Fault-Tolerant Routing". Proc. of the 21st Int. Symp. On Comp. Arch. Apr 1994, p.289-300
- [Ksn91] Konstanyinidou S, Snyder L. "Chaos Router: Architecture and Performance". Proc. of the 18th International Symposium on Computer Architecture, May 1991, pp.212-221
- [May93] May MD, Thompson PW, PH Welch Eds. "Networks, Routers and Transputers: Function, Performance and application". IOS Press 1993
- [Mer80] Merlin PM, Schweitzer PJ "Deadlock Avoidance in Store-and-Forward Networks-I: Store-and-Forward Deadlock" IEEE Trans. On Comm. V. Com-28, N.3, Mar 1980,345-354
- [NG92] Ni L, Glass C. "The Turn model for Adaptive Routing". Proc. of the 19th Intl. Symp. on Computer Architecture, IEEE Computer Society, May 1992, pp. 278-287
- [Pfi85] Pfister GF, Norton A. "Hot-Spot Contention and Combining in Multistage Interconnection Networks". IEEE Trans. On Computers. Vol. 34. N.10 Oct 1985.
- [PG+94] Pifarre GD et al. "Fully adaptive Minimal Deadlock Free Packet Routing in Hypercubes, Meshes and Other Networks: Algorithms and Simulations" IEEE Transactions on Parallel and Distributed Systems, V. 5, N.3, Mar 1994
- [Sni95] Snir M, Hochschild P, Frye DD, Gildea KJ "The communication software and parallel environment of the IBM SP2". IBM Systems Journal. Vol.34, N.2, pp. 205-221.
- [Val81] Valiant LG, Brebner GJ. "Universal Schemes for Parallel Communication". ACM STOC. Milwaukee 1981. pp. 263-277
- [Yan89] Yantchev JT, Jesshope CR. "Adaptive, Low Latency, Deadlock Free Packet Routing for Networks of Processors". IEE Proceedings. Vol 136, Pt.E, N.3; May 1989

Calculation of Lambda Modes of a Nuclear Reactor: A Parallel Implementation Using the Implicitly Restarted Arnoldi Method

Vicente Hernández, José E. Román, Antonio M. Vidal, and Vicent Vidal

Dept. Sistemas Informáticos y Computación, Universidad Politécnica de Valencia,
Camino de Vera, s/n, 46071 Valencia (Spain)
{vhernand,jroman,avidal,vvidal}@dsic.upv.es

Abstract. The objective of this work is to obtain the dominant λ -modes of a nuclear power reactor. This is a real generalized eigenvalue problem, which can be reduced to a standard one. The method used to solve it has been the Implicitly Restarted Arnoldi (IRA) method. Due to the dimensions of the matrices, a parallel approach has been proposed, implemented and ported to different platforms. This includes the development of a parallel iterative linear system solver. To obtain the best performance, care must be taken to exploit the structure of the matrices.

Keywords. Parallel computing, eigenproblems, lambda modes

1 Introduction

The generalized algebraic eigenvalue problem is a standard problem that frequently arises in many fields of science and engineering. In particular, it appears in approximations of differential operators eigenproblems. The application presented here is taken from nuclear engineering.

The analysis of the lambda modes are of great interest for reactor safety and modal analysis of neutron dynamical processes. In order to study the steady state neutron flux distribution inside a nuclear power reactor and the sub-critical modes responsible for the regional instabilities produced in the reactors, it is necessary to obtain the dominant λ -modes and their corresponding eigenfunctions.

The discretization of the problem leads to an algebraic eigensystem which can reach considerable sizes in real cases. The main aim of this work is to solve this problem by using appropriate numerical methods and introducing High Performance Computing techniques so that response time can be reduced to the minimum. In addition to this, other benefits can be achieved, as well. For example, greater problems can be faced and a better precision in the results can be attained.

This contribution is organized as follows. Section 2 is devoted to present how the algebraic generalized eigenvalue problem is derived from the neutron diffusion equation. In section 3, a short description of the matrices which arise

in this problem is done. Section 4 reviews the method used for the solution of the eigenproblem. Section 5 describes some implementation issues, whereas in section 6 the results are summarized. Finally, the main conclusions are exposed in section 7.

2 The Neutron Diffusion Equation

Reactor calculations are usually based on the multigroup neutron diffusion equation [12]. If this equation is modeled with two energy groups, then the problem we have to deal with is to find the eigenvalues and eigenfunctions of

$$\mathcal{L}\phi_i = \frac{1}{\lambda_i} \mathcal{M}\phi_i, \quad (1)$$

where

$$\mathcal{L} = \begin{bmatrix} -\nabla(D_1 \nabla) + \Sigma_{a1} + \Sigma_{12} & 0 \\ -\Sigma_{12} & -\nabla(D_2 \nabla) + \Sigma_{a2} \end{bmatrix},$$

$$\mathcal{M} = \begin{bmatrix} \nu_1 \Sigma_{f1} & \nu_2 \Sigma_{f2} \\ 0 & 0 \end{bmatrix} \quad \text{and} \quad \phi_i = \begin{bmatrix} \phi_{f,i} \\ \phi_{t,i} \end{bmatrix},$$

with the boundary conditions $\phi_i|_{\Gamma} = 0$, where Γ is the reactor border.

For a numerical treatment, this equation must be discretized in space. Nodal methods are extensively used in this case. These methods are based on approximations of the solution in each node in terms of an adequate base of functions, for example, Legendre polynomials [9]. It is assumed that the nuclear properties are constant in every cell. Finally, appropriate continuity conditions for fluxes and currents are imposed.

This process allows to transform the original system of partial differential equations into an algebraic large sparse generalized eigenvalue problem

$$L\psi_i = \frac{1}{\lambda_i} M\psi_i,$$

where L and M are matrices of order $2N$ with the following N -dimensional block structure

$$\begin{bmatrix} L_{11} & 0 \\ -L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} \psi_{1,i} \\ \psi_{2,i} \end{bmatrix} = \frac{1}{\lambda_i} \begin{bmatrix} M_{11} & M_{12} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \psi_{1,i} \\ \psi_{2,i} \end{bmatrix} \quad (2)$$

being L_{11} and L_{22} nonsingular sparse symmetric matrices, and M_{11} , M_{12} and L_{21} diagonal matrices. By eliminating $\psi_{2,i}$, we obtain the following N -dimensional non-symmetric standard eigenproblem

$$A\psi_{1,i} = \lambda_i \psi_{1,i},$$

where the matrix A is given by

$$A = L_{11}^{-1} (M_{11} + M_{12} L_{22}^{-1} L_{21}). \quad (3)$$

All the eigenvalues of this equation are real. We are only interested in calculating a few dominant ones with the corresponding eigenvectors.

This problem has been solved with numerical methods such as Subspace Iteration, [10], [11]. Here we present a more effective strategy.

3 Matrix Features

In order to validate the correctness of the implemented programs, two reactors have been chosen as test cases.

The first benchmark is the BIBLIS reactor [3], which is a pressure water reactor (PWR). Due to its characteristics, this reactor has been modeled in a bidimensional fashion with 1/4 symmetry. The nodalization scheme is shown in figure 1(a). The darkest cells represent the 23.1226 cm wide reflector whereas the other cells correspond to the reactor kernel, with a distance between nodes of 23.1226 cm as well. The kernel cells can be of up to 7 different materials.

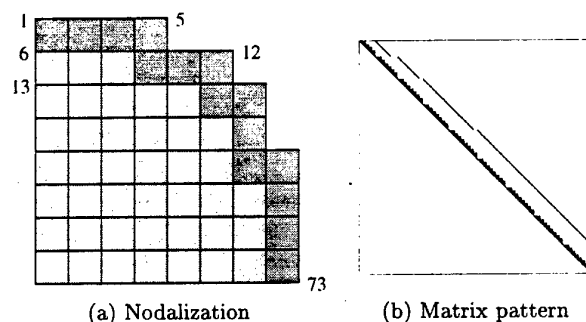


Fig. 1. The BIBLIS benchmark.

The numbers of the nodes follow a left-right top-down ordering, as shown in the figure. This leads to a staircase-like matrix pattern which can be seen in figure 1(b). Note that only the upper triangular part is stored. This pattern is identical for both L_{11} and L_{22} . In particular, the matrix pattern depicted in the figure corresponds to a space discretization using Legendre polynomials of 5th degree. This degree is directly related to the number of rows and columns associated to every node in the mesh. The dimensions of the matrices are shown in table 1.

The other reference case is the RINGHALS reactor [4], a real boiling water reactor (BWR). This reactor has been discretized three-dimensionally in 27 axial planes (25 for the fuel and 2 for the reflector). In its turn, each axial plane is divided in 15.275 cm \times 15.275 cm cells distributed as shown in figure 2(a). In this case, it is not possible to simplify the problem because the reactor does not have any symmetry by planes. Each of the 15600 cells has different neutronic properties. As expected, matrices arising from this nodalization scheme are much larger and have a much more regular structure (figure 2(b)).

Apart from symmetry and sparsity, the most remarkable feature of the matrices is bandedness. This aspect has been exploited in the implementation.

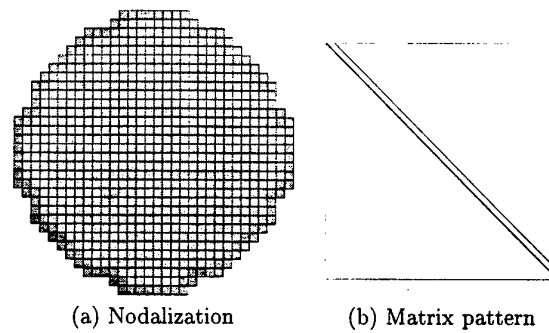


Fig. 2. The RINGHALS benchmark.

In table 1 several properties of the matrices are listed to give an idea of the magnitude of the problem. In this table, *dpol* is the degree of Legendre polynomials, *n* is the dimension of the matrix, *nz* is the number of non-zero elements stored the upper triangle, *bw* is the upper bandwidth, and *disp* is the percentage of non-zero values with respect to the whole matrix. Finally, the storage requirements for the values are given (*mem*) to emphasize this issue.

	dpol	n	nz	mem	bw	disp
Biblis	1	73	201	1.6 Kb	10	0.062
	2	219	1005	7.9 Kb	29	0.037
	3	438	2814	22 Kb	57	0.027
	4	730	6030	47 Kb	94	0.021
	5	1095	11055	86 Kb	140	0.017
Ringhals	1	20844	80849	0.6 Mb	773	0.00032
	2	83376	505938	3.9 Mb	3090	0.00013
	3	208440	1721200	13 Mb	7723	0.000074
	4	416880	4355110	33 Mb	15444	0.000048
	5	729540	9218685	70 Mb	27025	0.000033

Table 1. Several properties of the matrices.

4 Implicitly Restarted Arnoldi Method

As we only want to determine the dominant eigenvalues which define the reactor behaviour, we approach this partial eigenproblem with the Arnoldi method.

The Arnoldi method is a *Krylov subspace* or *orthogonal projection* method for extracting spectral information. We call

$$\mathcal{K}_k(A, v_0) = \text{span} \{v_0, Av_0, A^2v_0, \dots, A^{k-1}v_0\}$$

the k -th Krylov subspace corresponding to $A \in \mathbb{C}^{n \times n}$ and $v_0 \in \mathbb{C}^n$. The idea is to construct approximate eigenvectors in this subspace.

We define a k -step Arnoldi factorization of A as a relationship of the form

$$AV = VH + fe_k^T$$

where $V \in \mathbb{C}^{n \times k}$ has orthonormal columns, $V^H f = 0$, and $H \in \mathbb{C}^{k \times k}$ is upper Hessenberg with a non-negative sub-diagonal. The central idea behind this factorization is to construct eigenpairs of the large matrix A from the eigenpairs of the small matrix H .

In general, we would like the starting vector v_0 to be rich in the directions of the desired eigenvectors. In some sense, as we get a better idea of what the desired eigenvectors are, we would like to adaptively refine v_0 to be a linear combination of the approximate eigenvectors and restart the Arnoldi factorization with this new vector instead. A convenient and stable way to do this without explicitly computing a new Arnoldi factorization is given by the Implicitly Restarted Arnoldi (IRA) method, based on the implicitly shifted QR factorization [8].

The idea of the IRA method is to extend a k -step Arnoldi factorization

$$AV_k = V_k H_k + f_k e_k^T$$

to a $(k+p)$ -step Arnoldi factorization

$$AV_{k+p} = V_{k+p} H_{k+p} + f_{k+p} e_{k+p}^T.$$

Then p implicit shifts are applied to the factorization, resulting in the new factorization

$$AV_+ = V_+ H_+ + f_{k+p} e_{k+p}^T Q,$$

where $V_+ = V_{k+p} Q$, $H_+ = Q^H H_{k+p} Q$, and $Q = Q_1 Q_2 \cdots Q_p$, where Q_i is associated with factoring $(H - \sigma_i I) = Q_i R_i$. It turns out that the first $k-1$ entries of $e_{k+p} Q$ are zero, so that a new k -step Arnoldi factorization can be obtained by equating the first k columns on each side:

$$AV_k^+ = V_k^+ H_k^+ + f_k^+ e_k^T.$$

We can iterate the process of extending this new k -step factorization to a $(k+p)$ -step factorization, applying shifts, and condensing. The payoff is that every iteration implicitly applies a p^{th} degree polynomial in A to the initial vector v_0 . The roots of the polynomial are the p shifts that were applied to the factorization. Therefore, if we choose as the shifts σ_i eigenvalues that are "unwanted", we can effectively filter the starting vector v_0 so that it is rich in the direction of the "wanted" eigenvectors.

5 Implementation

This section describes some details of the implemented codes for a distributed memory environment. The Message Passing Interface (MPI), [2], has been used as the message passing layer so that portability is guaranteed.

5.1 Eigensolver Iteration

The Implicitly Restarted Arnoldi (IRA) method which has been used in this work is that implemented in the ARPACK [5] software package. This package contains a suit of codes for the solution of several types of eigenvalue related problems, including standard and generalized eigenproblems and singular value decompositions for both real and complex matrices. It implements the IRA method for non-symmetric matrices and the analogous Lanczos method for symmetric matrices.

In particular, the programs implemented for the calculation of the lambda modes make use of the parallel version of ARPACK [6], which is oriented to a SPMD/MIMD programming paradigm. This package uses a distribution of all the vectors involved in the algorithms by blocks among the available processors. The size of the block can be established by the user, thus allowing more flexibility for load balancing.

As well as in many other iterative methods packages, ARPACK subroutines are organized in a way that they offer a reverse communication interface to the user [1]. The primary aim of this scheme is to isolate the matrix-vector operations. Whenever the iterative method needs the result of an operation such as a matrix-vector product, it returns control to the user's subroutine that called it. After performing this operation, the user invokes the iterative method subroutine again.

The flexibility of this scheme gives the possibility of using various matrix storage formats as well as obtaining the eigenvalues and eigenvectors of a matrix for which an explicit form is not available. Indeed, the problem we are presenting here is a non-symmetric standard partial eigenproblem of an operator given by the expression (3), where A is not calculated explicitly.

The explicit construction of the inverses would imply the loss of sparsity properties, thus making the storage needs prohibitive. For this reason, the matrix-vector product needed in the Arnoldi process has to be calculated by performing the operations which appear in (3) one at a time. The necessary steps to compute $y = Ax$ are the following:

1. Calculate $w_1 = M_{11}x$.
2. Calculate $w_2 = L_{21}x$.
3. Solve the system $L_{22}w_3 = w_2$ for w_3 .
4. Calculate $w_4 = w_1 + M_{12}w_3$.
5. Solve the system $L_{11}y = w_4$ for y .

It has to be noted that the above matrix-vector products involve only diagonal matrices. Therefore, the most costly operations are the solution of linear systems of equations (steps 3 and 5).

5.2 Linear Systems of Equations

The resolution of the linear systems can be approached with iterative methods, such as the Conjugate Gradient [7]. These methods, in their turn, typically use the aforementioned reverse communication scheme.

For the parallel implementation, two basic operations have to be provided, namely the matrix-vector product and the dot product.

In this case, the matrix-vector product subroutine deals with sparse symmetric matrices (L_{ii}). The parallel implementation of this operation is described later. For the distributed dot product function, each processor can perform a dot product on the sub-vectors it has, and then perform a summation of all the partial results.

In order to accelerate the convergence, a Jacobi preconditioning scheme was used because of its good results and also because its parallelization is straightforward.

5.3 Parallel Matrix-Vector Product

The matrices involved in the systems of equations (L_{ii}) are symmetric, sparse and with their nonzero elements within a narrow band. This structure must be exploited for an optimal result. The storage scheme used has been Compressed Sparse Row containing only the upper triangle elements including the main diagonal.

A multiplication algorithm which exploits the symmetry can view the product $L_{ii}x = y$ as $(U + (L_{ii} - U))x = y$, where U is the stored part. Thus, the product can be written as the addition of two partial products,

$$\underbrace{Ux}_{y_1} + \underbrace{(L_{ii} - U)x}_{y_2} = y.$$

The algorithm for the first product can be row-oriented ($y_i = U^{(i)}x$) whereas the other must be column-oriented ($y = 0, y = y + x_i(A - U)_i = y + x_iU^{(i)T}$).

The parallel matrix-vector product has to be highly optimized in order to achieve good overall performance, since it is the most time-consuming operation. The matrices have been partitioned by blocks of rows conforming with the partitioning of the vectors. It has been implemented in three stages, one initial communication stage, parallel computation of intermediate results and finally another communication operation. This last stage is needed because only the upper triangular part of the symmetric matrices is stored. In the communication stages, only the minimum amount of information is exchanged and this is done synchronously by all the processors.

Figure 3 shows a scheme of the parallel matrix-vector product for the case of 4 processors. The shadowed part of the matrix corresponds to the elements stored in processor 1. Before one processor can carry out its partial product, a fragment of the vector operand owned by the neighbour processor is needed. Similarly, after the partial product is complete, part of the solution vector must be exchanged.

The detailed sequence of operations in processor i is the following:

1. Receive from processor $i + 1$ the necessary components of x . Also send to processor $i - 1$ the corresponding ones.

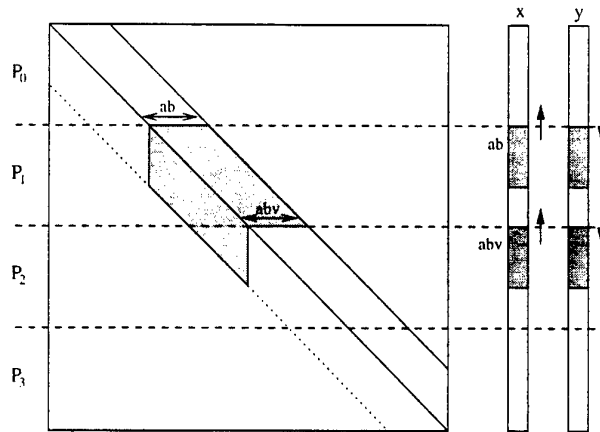


Fig. 3. Message passing scheme for the matrix-vector product.

2. Compute the partial result of y .
3. Send to processor $i + 1$ the fragment of y assigned to it. Get from processor $i - 1$ the part which must be stored in the local processor, as well.
4. Add the received block of y to the block calculated locally, in the appropriate positions.

The communication steps 1 and 3 are carried out synchronously by all the processors. The corresponding MPI primitives have been used in order to minimize problems such as network access contention.

6 Results

The platforms on which the code has been tested are a Sun Ultra Enterprise 4000 multiprocessor and a cluster of Pentium II processors at 300 MHz with 128 Mb of memory each connected with a Fast Ethernet network. The code has been ported to several other platforms as well.

Several experiments have shown that the most appropriate method for the solution of the linear systems in this particular problem is the Conjugate Gradient (CG) with Jacobi preconditioning. Other iterative methods have been tested, including BGC, BiCGStab, TFQMR and GMRES. In all the cases, the performance of these solvers has turned out to be worse than that of CG. Table 2 compares the average number of $L_{ii}x_k$ products and the time (in seconds) spent by the programs in the solution of the Ringhals benchmark ($d_{pol} = 1$) for each of the tested methods. Apart from the response time, it can be observed also in this table that CG is the solver which requires less memory. The time corresponding to eight processors is also included to show that the efficiency (E_p) is modified slightly.

Method	$L_{ii}x_k$	Time (p=1)	Time (p=8)	E_p (%)	Memory
CG	12	102.04	15.12	84	$5n$
BCG	22	174.32	24.84	88	$7n$
BiCGStab	15	129.99	19.03	85	$8n$
TFQMR	13	155.59	22.75	86	$11n$
GMRES(3)	16	210.79	30.49	86	$\sim 5n$

Table 2. Comparison between several iterative methods.

6.1 Adjustment of Parameters

When applied to this particular problem, the IRA method is a convergent process in all the studied cases. However, it is not an approximate method in the sense that the precision of the obtained approximate solution depends to a great extent on the tolerance demanded in the iterative process for the solution of linear systems of equations. Table 3 reflects the influence of the precision required in the Conjugate Gradient process ($\|L_{ii}\tilde{x} - b\|_2 < tol_{CG}$) in values such as average number of matrix-vector products ($L_{ii}x_k$), number of Arnoldi iterations (IRA) and precision of the obtained eigenvalue. It can be observed that the number of significant digits in the approximate solution λ_1 matches the required precision tol_{CG} and that, after a certain threshold (10^{-4}), the greater the tolerance, the worse the approximate solution is.

tol_{CG}	$L_{ii}x_k$	IRA	Ax	Time	λ_1	$\ Ax - \lambda x\ / \lambda $
10^{-9}	33	19	60	157.59	1.012189	0.000096
10^{-8}	29	19	60	139.33	1.012189	0.000096
10^{-7}	25	19	60	120.48	1.012189	0.000096
10^{-6}	21	19	60	102.16	1.012189	0.000097
10^{-5}	17	19	60	85.71	1.012189	0.000101
0.0001	13	19	60	67.16	1.012170	0.000354
0.001	9	23	72	57.47	1.011175	0.004809
0.01	5	38	117	58.14	1.003579	0.033075
0.1	2	14	44	12.04	0.801034	0.297213

Table 3. Influence of the CG tolerance in the precision of the final result.

In conclusion, the tolerance to be demanded in both the IRA and CG methods should be of the same magnitude. In our case, it is sufficient to choose 10^{-4} and 10^{-5} , respectively, since input data are perturbed by inherent errors of the order of 10^{-6} .

Another parameter to be considered is the number of columns of V (ncv), that is, the maximum dimension of the Arnoldi basis before restart. This value has a great influence in the effectiveness of the IRA method, either in its computational cost as well as in the memory requirements. If ncv takes a great value, the

computational cost per iteration and the storage requirements can be prohibitive. However, a small value can imply that the constructed Krylov subspace contains too few information and, consequently, the process needs too many iterations until convergence is achieved. Some experiments reveal that, in this particular application, a value of 2 or 3 times the number of desired eigenvalues gives good results.

Figure 4 shows the number of matrix-vector operations (Ax_k) necessary to achieve convergence for different values of ncv . The experiment has been repeated for a value of desired eigenvalues (nev) ranging from 1 to 6. In the case of only one eigenvalue, it is clear that a small subspace dimension makes the convergence much slower. On the other hand, incrementing ncv beyond some value is useless with respect to convergence while increasing the memory requirements.

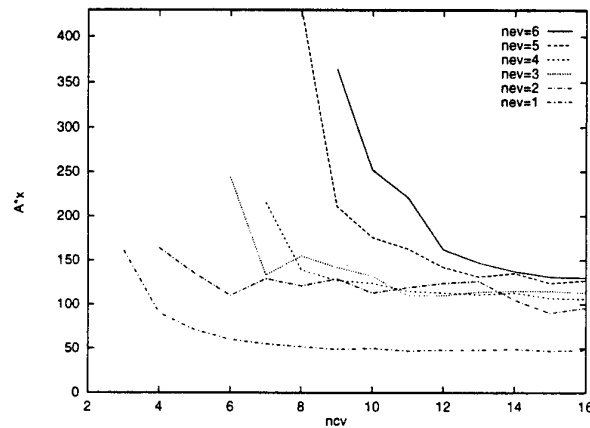


Fig. 4. Influence of ncv in the convergence of the algorithm.

In the other five cases, one can observe a similar behaviour. However, the lines are mixed, mainly in the next three eigenvalues. This is caused by the known fact that clusters of eigenvalues affect the convergence. Figure 5 shows the convergence history of the first 6 eigenvalues. In this graphic it can be appreciated that eigenvalues 2, 3 and 4 converge nearly at the same time, because they are very close to each other (see table 4).

6.2 Speed-up and Efficiency

In figure 6 we give the performance of the parallel eigensolver measured in the Sun multiprocessor. The graphics show the execution time (in seconds), speed-up and efficiency with up to eight processors for each of the five matrices corresponding to the Ringhals benchmark. The Biblis reactor has not been considered for measurement of parallel performance because of its small size.

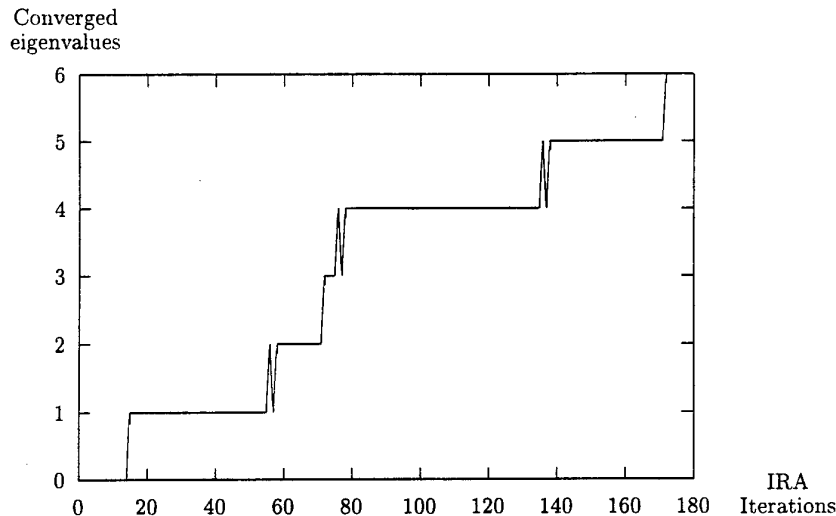


Fig. 5. Convergence history of the first 6 dominant eigenvalues of the Ringhals case (dpol=2).

	λ_i	$\ Ax - \lambda_i x\ / \lambda_i $
λ_1	1.012190	0.000038
λ_2	1.003791	0.000042
λ_3	1.003164	0.000043
λ_4	1.000775	0.000036
λ_5	0.995716	0.000049
λ_6	0.993402	0.000096

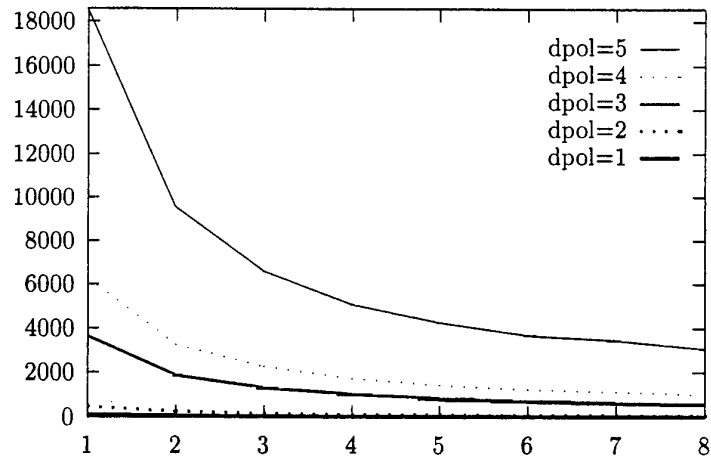
Table 4. Dominant eigenvalues of the Ringhals benchmark (dpol=2).

In these graphics, it can be seen that the speed-up increases almost linearly with the number of processors. The efficiency does not fall below 75% in any case. We can expect these scalability properties to maintain with more processors.

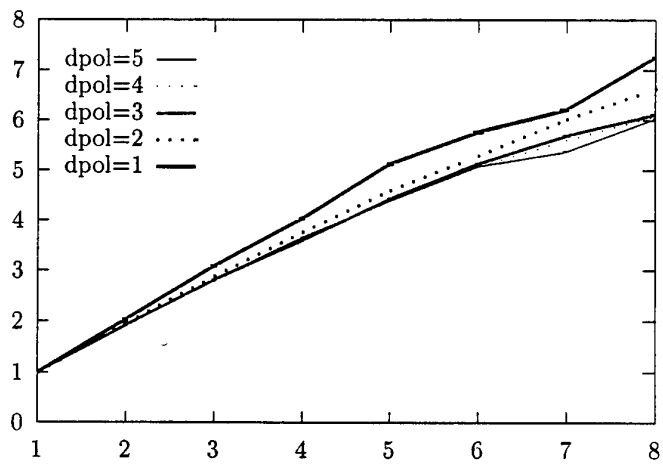
The discretization with polynomials of 2nd degree gives a reasonably accurate approximation for the Ringhals benchmark. In this case, the response time with 8 processors is about 70 seconds.

In the case of the cluster of Pentiums, the efficiency reduces considerably as a consequence of a much slower communication system. However, the results are quite acceptable. Figure 7 shows a plot of the efficiency attained for the five cases of the Ringhals benchmark. It can be observed that for the biggest matrix, the efficiency is always greater than 50% even when using 8 computers, corresponding to a speed-up of 4.

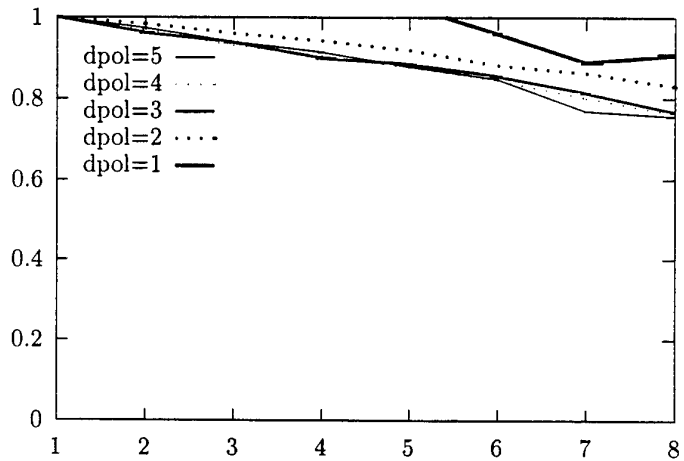
Another additional advantage in the case of personal computers must be emphasized: the utilization of the memory resources. The most complex cases (Ringhals with $dpol = 4$ and $dpol = 5$) can not be run on a single computer



(a) Execution time (in seconds)



(b) Speed-up



(c) Efficiency

Fig. 6. Graphics of performance of the eigensolver.

because of the memory requirements. Note that the memory size of the running process can be more than 250 Mb. The parallel approach gives the possibility to share the memory of several computers to solve a single problem.

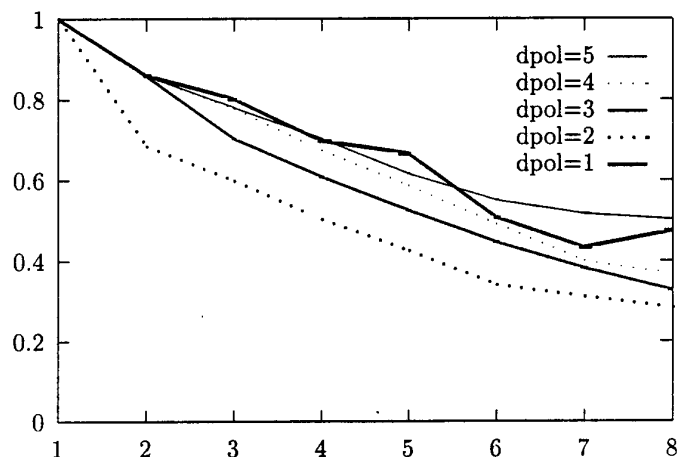


Fig. 7. Efficiency obtained in a cluster of PC's.

7 Conclusions

In this work, we have developed a parallel implementation of the Implicitly Restarted Arnoldi method for the problem of obtaining the λ -modes of a nuclear reactor. This parallel implementation can reduce the response time significantly, specially in complex realistic cases. The nature of the problem has forced to implement a parallel iterative linear system solver as a part of the solution process. With regard to scalability, the experiments with up to 8 processors have shown a good behaviour of the eigensolver.

Apart from the gain in computing time, another advantage of the parallel implementation is the possibility to cope with larger problems using inexpensive platforms with limited physical memory such as networks of personal computers. With this approach, the total demanded memory is evenly distributed among all the available processors.

The following working lines are being considered for further refinement of the programs:

- Store the solution vectors of the linear systems to use them as initial solution estimations in subsequent IRA iterations.
- Begin the Arnoldi process taking as initial estimated solution an extrapolation of the solution of the previous order problem.

- Use a more appropriate numbering scheme for the nodes of the grid in order to reduce the bandwidth of the matrices and, consequently, reduce the size of the messages exchanged between processors.
- Implement a parallel direct solver for the linear systems of equations, instead of using an iterative method. This should be combined with the reduction of bandwidth so that the fill-in is minimized.
- Consider the more general case of having G energy groups in the neutron diffusion equation, instead of only 2. In this case, it would probably be more effective to approach the problem as a generalized eigensystem.
- Consider also the general dynamic problem. In this case, the solution is time-dependent and a convenient way of updating it would be of interest.

References

1. J. Dongarra, V. Eijkhout, and A. Kalhan. Reverse communication interface for linear algebra templates for iterative methods. Technical Report UT-CS-95-291, Department of Computer Science, University of Tennessee, May 1995.
2. MPI Forum. MPI: a message-passing interface standard. *International Journal of Supercomputer Applications and High Performance Computing*, 8(3/4):159-416, Fall-Winter 1994.
3. A. Hébert. Application of the Hermite Method for Finite Element Reactor Calculations. *Nuclear Science and Engineering*, 91:34-58, 1985.
4. T. Lefvert. RINGHALS1 Stability Benchmark - Final Report. Technical Report NEA/NSC/DOC(96)22, November 1996.
5. R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK USERS GUIDE: Solution of Large Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods*. SIAM, Philadelphia, PA, 1998.
6. K. J. Maschhoff and D. C. Sorensen. PARPACK: An Efficient Portable Large Scale Eigenvalue Package for Distributed Memory Parallel Architectures. *Lecture Notes in Computer Science*, 1184:478-486, 1996.
7. Y. Saad. SPARSKIT: a basic tool kit for sparse matrix computation. RIACS Technical Report 90.20, NASA Ames Research Center, 1990 (updated 1991).
8. Danny C. Sorensen. Implicitly Restarted Arnoldi/Lanczos Methods For Large Scale Eigenvalue Calculations. Technical Report TR-96-40, Institute for Computer Applications in Science and Engineering, May 1996.
9. G. Verdú, J. L. Muñoz-Cobo, C. Pereira, and D. Ginestar. Lambda Modes of the Neutron-Diffusion Equation. Application to BWRs Out-of-Phase Instabilities. *Annals of Nuclear Energy*, 7(20):477-501, 1993.
10. V. Vidal, J. Garayoa, G. Verdú, and D. Ginestar. Optimization of the Subspace Iteration Method for the Lambda Modes Determination of a Nuclear Power Reactor. *Journal of Nuclear Science and Technology*, 34(9):929-947, September 1997.
11. V. Vidal, G. Verdú, D. Ginestar, and J. L. Muñoz-Cobo. Variational Acceleration for Subspace Iteration Method. Application to Nuclear Power Reactors. *International Journal for Numerical Methods in Engineering*, 41:391-407, 1998.
12. J. R. Weston and M. Stacey. *Space-Time Nuclear Reactor Kinetics*. Academic Press, New York, 1969.

Stochastic Control of the Scalable High Performance Distributed Computations

Zdzislaw Onderka

Institute of Computer Science, Jagiellonian University,
Nawojki 11, 30-072 Krakw, Poland
onderka@ii.uj.edu.pl, www.ii.uj.edu.pl/zpi/onderka

Abstract. The Markov Decision Control Problem of optimal task distribution in computer network is formulated. The *open-loop* and *closed-loop* control strategies based on the stochastic forecast are presented. Numerical tests for the mixed FE/FD scheme performed on the heterogeneus network exhibit the behavior of different strategies for various size of data to be processed.

Keywords: stochastic control, distributed computation, optimal task distribution, Markov Decision Process

1 Master-Slave Application and Heterogeneous Network Model

The *heterogeneous computer network and distributed application* is a tuple $\langle H, A, F, G_A \rangle$ ([8]), where:

- $H = \{M_0, \dots, M_m\}$ is a set of $m + 1$ distinct machines which may communicate with any other one, may have a different architecture (sequential, vector, parallel etc.) and performance parameters for each machine may vary in time. We assume that the computer network is *nondecomposable* [1].
- A is a *master-slave* application, which consists of sequential part t_0 (*master task*) and distributed part of N identical tasks (*slaves*) $V_D = \{t_1, \dots, t_N\}$. Additionally, we assume that tasks are constrained to be *atomic*.
- $G_A = (V, E)$, is a *scheduled task graph* for the application A [1]; $E \subseteq V \times V$, $V = V_D \cup \{t_0\}$, $(t_i, t_k) \in E$ iff t_i must execute before the t_k due to data dependence or task synchronization requirements of A . Without loss of generality, we assume that G_A has unique entry and exit nodes identified with the task t_0 , and for any task t_i there is a path from t_0 to t_i and a path from t_i to t_0 .
- F is a mapping $F : V \rightarrow H$ which defines the G_A and which in case of the dynamic tasks allocation is defined as a sequence of partial mappings, $\{F_n\}_{n=0,1,\dots}$, i.e.

$$\bigcup_n F_n^{-1}(M_i) = F^{-1}(M_i) \quad \forall M_i \in H \quad F^{-1}(M_0) = t_0 \quad (1)$$

2 Estimation of State of Background Workload

The considered network H is composed of the computers, which resources are shared between several users (*multi-user systems*). Therefore, processes may appear randomly on each machine $M \in H$, so the background load of M and background load of the communication network are changing in time and their changes have significant influence

on the large applications efficiency computed in the distributed network H . For that reason the background load of the machine and communication network is allowed to be estimated in the stochastic way. The advantage of such model is opportunity to define the functional dependencies between system performance parameters and workload parameters which gives us a knowledge about future evaluation of those parameters. Only the stochastic model for the background load of the network H was described in this paper considering the identical formal description for communication network load.

Let $load_j \in \mathcal{R}_+$ be the additional background load for the machine $M_j \in H$. Each value $load_j$ corresponds with the *execution slowing down parameter* $\eta_j \in \mathcal{R}_+$ (defined in [7][8]) (i.e. $\bar{T}_t^j = (1 + \eta_j) \cdot T_t^j$, where \bar{T}_t^j and T_t^j are the pattern task execution times with and without additional background load respectively). So, further we will consider the bijection $\psi_j : \mathcal{R}_+ \ni load_j \rightarrow \eta_j \in \mathcal{R}_+$.

Let consider Θ_j and Γ_j two ordered sets of thresholds:

$$\Theta_j = \{\theta_j(i) \in \mathcal{R}_+, i = 0, \dots, K_j, K_j \in \mathcal{N} : \forall i = 0, \dots, K_j - 1 \theta_j(i) < \theta_j(i + 1)\} \quad (2)$$

$$\Gamma_j = \{\eta_j(i) \in \mathcal{R}_+, i = 0, \dots, K_j, K_j \in \mathcal{N} : \forall i = 0, \dots, K_j - 1 \eta_j(i) < \eta_j(i + 1)\} \quad (3)$$

first one for the physical measured performance parameters (like $load_j$) and the second one for the slowing down parameters, and such that $\forall M_j \in H \forall i = 0, \dots, K_j \psi_j(\theta_j(i)) = \eta_j(i)$. So, $\forall M_j \in H$ we obtained the set $S_j = \{0, \dots, K_j\}$, which will be called the *set of states of the background load*.

Let define the mapping $\Psi_j : \mathcal{R}_+ \rightarrow S_j$ as:

$$\Psi_j(\eta_j) = \begin{cases} 0 & \text{dla } \eta_j(0) \leq \eta_j < \eta_j(1), \\ 1 & \text{dla } \eta_j(1) \leq \eta_j < \eta_j(2), \\ \vdots & \vdots \\ K & \text{dla } \eta_j(K) \leq \eta_j < \infty \end{cases} \quad (4)$$

where $\forall j \eta_j = \psi_j(load_j)$.

The machine $M_j \in H$ is in the state $i \in \{0, \dots, K_j\}$ of the background load *iff* $\Psi_j \circ \psi_j(load_j) = i$. Moreover, $\forall M_j \in H$ let define bijection $\varphi_1^j : S_j \ni i \rightarrow \eta_j(i) \in \Gamma_j$.

3 Model of Background Workload

As the *model of background load* we assume a tuple $\langle \wp, \{X_n\}_{n=0,1,\dots}, \mathcal{P}, \Gamma, \Theta, S, \varphi_1 \rangle$ ([8]), where:

- \wp is a tuple $\langle \wp_1, \dots, \wp_m \rangle$ where $\forall j = 1, \dots, m \wp_j = (\Omega, \mathfrak{F}, P^j)$ is a probability space where Ω is a set of events that M admits some average value of the execution slowing down parameter in the Δ_n for some n , identical for each M_j ([7]). In consequence each event corresponds with a unique state from S .

- $\{X_n\}_{n=0,1,\dots}$ is a tuple $\langle \{X_n^1\}_{n=0,1,\dots}, \dots, \{X_n^m\}_{n=0,1,\dots} \rangle$ of nonstationary discrete stochastic Markov chains ([7]) which describe dynamic behavior of state of background load for each $M_j \in H$ in time. The dynamics is given by:

$$P^k(X_{n+1}^k = j | X_n^k = i) = P^k(X_{n+1}^k = j | X_0^k = i_0, \dots, X_{n-1}^k = i_{n-1}, X_n^k = i) \quad (5)$$

Each $X_n^j : \Omega \rightarrow S$ corresponds to the time interval $\Delta_n, n = 0, 1, \dots$

- \mathcal{P} is a tuple $\langle \mathcal{P}^1, \dots, \mathcal{P}^m \rangle$ where $\mathcal{P}^k = \{P_n^k\}_{n=0,1,\dots}$ is a sequence of Markov transition matrices. The $p_{ij}^k(n) = P_n^k(X_{n+1}^k = j | X_n^k = i)$ is a probability that process will be in state j of background load at the step $n+1$ provided that it is in state i at step n for the machine M_k . If $\Pi(n) = (\Pi^1(n), \dots, \Pi^m(n))$ denotes the vector of the state probability distributions at the step n for the network H and $\Pi^j(\mu) = \beta_\mu = (\beta_\mu^1, \dots, \beta_\mu^m)$ is the initial distribution then the consecutive distributions may be evaluated according to the formula [6]: $\Pi^j(n) = \Pi^j(n-1) \cdot P_n^j, n > \mu$.
- Γ is a tuple $\langle \Gamma_1, \dots, \Gamma_m \rangle$ where Γ_j is the following set of thresholds for the slowing down parameters:

$$\Gamma_j = \{\eta_j(i) \in \mathcal{R}_+, i = 0, \dots, K, : \forall i = 0, \dots, K-1 \ \eta_j(i) < \eta_j(i+1)\} \quad (6)$$

- Θ is a tuple $\langle \Theta_1, \dots, \Theta_m \rangle$ where Θ_j is the following set of thresholds for the physical measured performance parameters (like $load_j$) for the machine M_j :

$$\Theta_j = \{\theta_j(i) \in \mathcal{R}_+, i = 0, \dots, K, : \forall i = 0, \dots, K-1 \ \theta_j(i) < \theta_j(i+1)\} \quad (7)$$

Moreover, $\forall j = 1, \dots, m \ \forall i = 0, 1, \dots, K \ \psi_j : \Theta_j \ni \theta_j(i) \rightarrow \eta_j(i) \in \Gamma_j$:

- S is a tuple $\langle S_1, \dots, S_m \rangle$ where S_j is the set of states of background load for the machine $M_j \in H$ and $card(S_j) = card(\Gamma_j) \ \forall j$.
- φ_1 is a tuple $\langle \varphi_1^1, \dots, \varphi_1^m \rangle$ and $\forall j = 1, \dots, m \ \varphi_1^j$ is a mapping $\varphi_1^j : S_j \rightarrow \Gamma_j$.

Having the stochastic processes $\{X_n^j\}_{n=0,1,\dots}$ which describes dynamic behavior of the background load of $M_j \in H$ for $j = 1, \dots, m$ and bijections φ_1 and $\varphi_2 = \langle \varphi_2^1, \dots, \varphi_2^m \rangle$ where $\varphi_2^j : \Gamma_j \rightarrow \mathcal{R}_+, \varphi_2^j(\eta_j) = (1 + \eta_j) \cdot T_t^j$ then we may define the new stochastic processes $\{\bar{T}_t^j(n)\}_{n=0,1,\dots}$ for $M_j \in H, j = 1, \dots, m$ describing the dynamic behavior of the times required to execute the single task $t \in V_D$ on the machine M_j with the background load and $\forall j \in \{1, \dots, m\}, n = 0, 1, \dots$

$$\bar{T}_t^j(n) = \varphi_2^j \circ \varphi_1^j(X_n^j) = (1 + \eta_j(X_n^j)) \cdot T_t^j \quad (8)$$

4 Stochastic Control Based on Background Load Model

The *control policy* u is the sequence $\{F_n\}_{n=\mu, \mu+1, \dots}$ where the subscript $\mu \geq 0$ denotes the starting time epoch τ_μ (the time step of the nonstationary stochastic process describing dynamic behavior of the background load) of the realized policy. The policy $u \in U_\mu$ is an *admissible policy* if

$$\begin{aligned} & \exists q \in (0, 1], \ \forall j = 1, \dots, m \ \forall n = \mu, \mu+1, \dots \\ & q \leq P_n^j \left(\left(\bar{T}_t^j(n) \cdot card(F_n^{-1}(M_j)) \right) < \Delta_n \right) \leq 1 \end{aligned} \quad (9)$$

where $P(\cdot)$ denotes the probability.

Let $C_n(\beta_\mu, s, u)$ denotes the *cost function for the step* $n = \mu, \mu + 1, \dots$ of the realized policy $u \in U_\mu$ (consequently, that is the cost of the function F_n applied in the period $[\tau_n, \tau_{n+1})$, $n = \mu, \mu + 1, \dots$), defined for the initial distribution $\beta_\mu = (\beta_\mu^1, \dots, \beta_\mu^m)$ of the state of background load in the time epoch τ_μ and achieved state $s = (s^1, \dots, s^m)$, $s^j \in S_j$ in τ_n . It is defined as:

$$C_n(\beta_\mu, s, u) = \max_{M_j \in H} (c^j(n, s^j, u)) \quad (10)$$

where $c^j(n, s^j, u)$ is an *immediate cost function* for machine $M_j \in H$ that is in the state $s^j \in S_j$. Also, we assume that $\forall u \in U_\mu, \forall \mu$ and $\tau_n \geq \tau_\mu$, and for any initial distribution β_μ^j and any state in the time epoch τ_n for M_j there exists an expectation value $E_{\beta_\mu^j}(c^j(n, s^j, u))$ [3,4,6].

Let $C^Z(\beta_\mu, u)$ be the *finite horizon cost for the network H* related to the policy $u \in U_\mu$ provided in the time period $[\mu, Z], (Z < \infty)$. It is defined as:

$$C^Z(\beta_\mu, u) = E_{\beta_\mu} \left(\sum_{n=\mu}^{\mu+Z-1} C_n(\beta_\mu, s, u) \right) \quad (11)$$

where $\beta_\mu = (\beta_\mu^1, \dots, \beta_\mu^m)$.

We define the control problem (see [4,7,8]) as:

Problem 1. Find a policy $u \in U_\mu$ that minimizes $C^Z(\beta_\mu, u)$ over the whole U_μ .

5 MDP Based Stochastic Control

The following control problem is based on the theory of Markov Decision Process (MDP) [3, 4]. Let $\mathcal{T} = \{\tau_\mu, \dots, \tau_{\mu+Z-1}\}$ is a set of decision epochs corresponding to the beginning of the period Δ_n , $n = \mu, \dots$ and $Z \in \mathcal{N}$, $Z < \infty$. We assume that the last decision is made at $\tau_{\mu+Z-1}$.

In each decision epoch the *decision agent* receives the state $s = (s^1, \dots, s^m) \in S$ of network H , in which may choose an action $a = (a^1, \dots, a^m) \in A_s = A_{s^1}^1 \times \dots \times A_{s^m}^m$, where $A_{s^k}^k$ is the discrete set of actions which may be chosen if the state of machine M_k is s^k . Moreover, we assume the actions are chosen in deterministic way. Assume that $\mathcal{A} = \bigcup_s A_s$ and $\mathcal{A} = \mathcal{A}^1 \times \dots \times \mathcal{A}^m$ where $\mathcal{A}^k = \bigcup_{s^k \in S_k} A_{s^k}^k$. As a result of choosing the action $a^k \in A_{s^k}^k$ in the state s^k at the decision epoch τ_i is an *immediate cost function* $c^k(i, s^k, u)$ ($u \in U_\mu$) and the state of machine M_k in the decision epoch τ_{i+1} is determined by the probability distribution $p_{\tau_i}^k(\cdot | s^k, a^k)$ and $\sum_{j \in S} p_{\tau_i}^k(j | s^k, a^k) = 1$.

A *decision rule* prescribes a procedure for action selection in each state and decision epoch τ_i . It is defined as a function $d_\tau : S \ni s \rightarrow d_\tau(s) \in A_s$, where if $s = (s^1, \dots, s^m)$ then $d_\tau(s) = (d_\tau^1(s^1), \dots, d_\tau^m(s^m))$. The *control policy* is defined as the sequence $u = (d_0, d_1, \dots, d_{Z-1})$. We will use the finite horizon Markov deterministic policies [3.4.6] in order to control the execution of the distributed application. It means, that each decision rule depends on previous states of the machines whole workload and selected actions in these states, and each action is chosen with certainty. The whole workload for the machine will be understood as the common load of background and load derived from the execution of the task belonging to an application A .

The *admissible policy* u is defined by the formula (9) where

$$\{d_n(s_n)\}_{n=0,1,\dots,Z-1} = \{a_n\}_{n=\mu,\mu+1,\dots} = \{F_n\}_{n=\mu,\mu+1,\dots} \quad (12)$$

The sample space Ω^H for the network H has the form:

$$\Omega^H = S \times \mathcal{A} \times S \times \dots \times \mathcal{A} \times S = \{S \times \mathcal{A}\}^{Z-1} \times S \quad (13)$$

and the elementary event $\omega^H \in \Omega^H$ is a sequence

$$\Omega^H \ni \omega^H = (s_0, a_0, s_1, a_1, \dots, a_{Z-1}, s_Z) \quad (14)$$

$$\forall n \quad s_n = (s_n^1, \dots, s_n^m), \quad a_n = (a_n^1, \dots, a_n^m)$$

The random variables $\bar{X}_i(\omega^H)$, $\omega^H \in \Omega^H$, $i = 0, 1, \dots$ are defined as $\bar{X}_i : \Omega^H \rightarrow S$ and $\bar{X}_i(\omega^H) = s_i$, where i corresponds to the decision epoch τ_i (and time period Δ_i). The sequence of random variables $\bar{X}_i^k(\omega)$, $\omega \in \Omega$, $i = 0, 1, \dots$ are defined as $\bar{X}_i^k : \Omega \rightarrow S^k$ and $\bar{X}_i^k(\omega) = s_i^k$ where $s_i^k \in S^k$ is the state of the common load of background and load derived from the execution of the task belonging to an application A (the whole workload).

If we denote by h_i^k the history for the machine $M_k \in H$ i.e.

$$h_i^k = (\bar{X}_0^k = s_0^k, Y_0^k = a_0^k, \dots, Y_{i-1}^k = a_{i-1}^k, \bar{X}_i^k = s_i^k), \quad \mu \leq \tau_i \leq \mu + Z - 1 \quad (15)$$

then the dynamic behavior of the state of the whole workload of machine M_k is given by:

$$\begin{aligned} P_u^k(\bar{X}_{n+1}^k = j | h_n^k = (h_{n-1}^k, a_{n-1}^k, i), Y_n^k = a_n^k) = \\ P_u^k(\bar{X}_{n+1}^k = j | \bar{X}_n^k = i, Y_n^k = a_n^k) = p_{\tau_n}^k(j | i, a_n^k) \end{aligned} \quad (16)$$

and $h_i = (h_i^1, \dots, h_i^m)$ and $Y_n(\omega^H) = a_n \in A_{s_n}$, $\bar{X}_n(\omega^H) = s_n$.

The state probabilities $\bar{\Pi}^k(\mu + i + 1)$ for random variable $\bar{X}_{\mu+i+1}^k$ at the decision epoch $\tau_{\mu+i+1}$ can be evaluated by:

$$\bar{\Pi}^k(\mu + i + 1) = \sum_{j \in S^k} p_{\tau_{\mu+i}}^k(j | s_j^k, a_j^k) \cdot \bar{\Pi}^k(\mu + i) \quad (17)$$

where $\bar{\Pi}^k(\mu + i)$ is the probability distribution at the decision epoch $\tau_{\mu+i}$.

We have no probabilities $p_{\tau_{\mu+i}}^k(j | s_j^k, a_j^k)$ so, the computation of probability distribution $\bar{\Pi}^k(\mu + i)$ will be based on the probability distribution $\Pi^k(\mu + i)$ of the state of background load. Let sign $r_i = \eta(i + 1) - \eta(i)$ and δ is the load derived from the task of application A , and $\alpha_i = \frac{\delta}{r_i}$. Let us assume that the load δ is less then r_i , then the evaluation of the probability distribution $\bar{\Pi}^k(\mu + i)$ is as follows (in matrix form) ([8]):

$$(\bar{\Pi}^k(n))^T = B \cdot (\Pi^k(n))^T = B \cdot (P^k(n) \cdot \Pi^k(n-1))^T \quad (18)$$

The superscript T denotes the matrix transposition and matrix B has the following form:

$$\begin{bmatrix} 1 - \alpha_1 & 0 & 0 & 0 & \dots \\ \alpha_1 & 1 - \alpha_2 & 0 & 0 & \dots \\ 0 & \alpha_2 & 1 - \alpha_3 & 0 & \dots \\ \vdots & & & & \\ 0 & \dots & \alpha_{K-1} & 1 \end{bmatrix} \quad (19)$$

The finite horizon cost for network H for the deterministic Markov policy $u \in U_\mu$ and for the finite horizon Z is defined as:

$$C^Z(\beta_\mu, u) = E_{\beta_\mu}^u \left(\sum_{n=\mu}^{\mu+Z-1} C_n(n, \bar{X}_n, d_n(h_n)) \right) \quad (20)$$

where $d_n(h_n) = Y_n$, $h_n = (h_{n-1}, a_{n-1}, s)$, and $P_n^k(Y_n^k = a_n^k | s_n^k) = 1$ and $\beta_\mu = (\beta_\mu^1, \dots, \beta_\mu^m)$.

We define the control problem **CP2** ([8,9]) as:

Problem 2. Find a policy $u \in U_\mu$ that minimizes $C^Z(\beta_\mu, u)$ over the whole U_μ .

The existence of the optimal Markov deterministic policy is guaranteed by the following theorem (see. [3,8]):

Theorem 3. Assume S is finite or countable. Then if

1. $\forall s$ the set of actions A_s is finite, or
2. A_s is compact, the cost function $C_n(\beta_\mu, s, a)$ for the step n of the realized policy is continuous in $a \forall s \in S$, and $\exists C < \infty$ for which $C_n(\beta_\mu, s, a) \leq C \forall a \in A_s, s \in S$, and $p_n^k(j | s_n^k, a_n^k)$ is continuous in $a_n^k \forall j \in S_k$ and $s_n^k \in S^k$ and $n = \mu, \dots, Z-1$

then there exists a deterministic Markovian policy which is optimal.

6 The Policies of Tasks Distributions

We will consider policies which are leading to the fastest execution of the application A . They generally fall in two groups: the group of deterministic policies and the group of stochastic policies based on the described Markov models.

The following policies from the first group: *single task*, *multiple task*, *dynamic single distribution*, *stationary* are described in detail in [7, 8].

In order to execute the group of stochastic policies, we introduce the following classes of agents: *state agents* - that monitors the load in time and prepares the state forecast for each machine in H ; *decision making agents*, that are involved in establishing task allocation policy based on actual and forecasted average state of each machine in H (or forecasted computation time of task) [7, 8]. Below there are presented two algorithms belonging to the group of stochastic policies: first based on the background load model (*open loop control*) (§3) and the second one based on the MDP model (*closed loop control*) (§4) [8].

6.1 Open Loop Control

The following algorithm is based on the average state of the background load forecast generated only once in the starting time epoch τ_μ for each machine in H . More precisely, this policy rely on the computation of the distributions $\Pi_i^k(n)$ having the initial distributions of the state of background load β_μ^k for each machine $M_k \in H$ recursively according to the formula:

$$\Pi_i^k(n) = \sum_{j \in S} p_{ji}^k(n-1) \cdot \Pi_j^k(n-1), \quad \tau_n > \tau_\mu \quad (21)$$

Now, having the vector of the state probabilities for each time epoch we may compute the expected execution times of one task on each machine in each time step which leads to the computation of *power coefficients* λ_j for all machines M_j as well as *power coefficients* Λ for the network H (defined as ([7,8]): $\lambda_j = \frac{1}{T_j}$, $\Lambda = \sum_{j=1}^m \lambda_j$)

The consecutive decision rule (or function F_n) allocates the subsets of tasks $F^{-1}(M_k)$ for which cardinal numbers are proportional to the machines expected *power coefficients* for the $n = \mu, \mu + 1, \dots, \mu + Z - 1$.

Algorithm 1

1. $n = \mu$;
 $V(n) = V_D$;
2. FOR(each $M_j \in H$)
 Let $\lambda_j(n) = \frac{1}{E_{\beta_\mu^j}(T_t^{M_j}(n))}$;

 $C = \sum_{M_j \in H} [\Delta_n \cdot \lambda_j(n)]$;
3. IF($C < \text{card}(V)$)
 { FOR(each $M_j \in H$)
 Let $F_n^{-1}(M_j) \subset V(n)$ such that $\text{card}(F_n^{-1}(M_j)) = [\Delta_n \cdot \lambda_j(n)]$;
 $V(n+1) = V(n) \setminus \bigcup_j F_n^{-1}(M_j)$;
 $n = n + 1$;
 GO TO 2;
 }

 ELSE /* last step */
 { $\Lambda = \sum_{M_j} \lambda_j(n)$;
 FOR(each $M_j \in H$)
 $\varrho_j = \frac{\lambda_j(n)}{\Lambda}$;
 FOR(each $M_j \in H$)
 Let $W_j = F_n^{-1}(M_j)$ such that the set $\left\{ \left| \frac{\text{card}(F_n^{-1}(M_j))}{\text{card}(V(n))} - \varrho_j \right| \right\}$
 is minimal over $\{W_j\}_{M_j \in H}$ and $\bigcup_j W_j = V(n)$;
 }
4. /*** start execution for the horizon Z ***/

The expected time of a single task execution on the machine M_k in each next time step $n = \mu, \mu + 1 \dots$ is as follows([8]):

$$E_{\beta_\mu^k}(\bar{T}_t^k(n)) = E_{\beta_\mu^k}(\varphi_2^k \circ \varphi_1^k(X_n^k)) =$$

$$\begin{cases} (1 + \eta_k(j)) \cdot T_t^k & \text{dla } n = \mu \\ \sum_{j=0}^K (\Pi_j^k(n) \cdot (1 + \eta_k(j)) \cdot T_t^k) & \text{dla } n > \mu \end{cases} \quad (22)$$

but the expected cost for one step $C_n(\beta_\mu, s_n, u)$ of the realized policy $u = \{F_n\}_{n=\mu, \mu+1, \dots}$ is as follows ([8]):

$$C_n(\beta_\mu, s_n, u) = \quad (23)$$

$$\begin{cases} \max_{F_n(M_k)} \{v^k(n) \cdot (1 + \eta_k(j)) \cdot T_t^k\} & \text{for } n = \mu \\ \max_{F_n(M_k)} \{v^k(n) \cdot \sum_{j=0}^{K^k} (\Pi_j^k(n) \cdot (1 + \eta_k(j)) \cdot T_t^k)\} & \text{for } n > \mu \end{cases}$$

where $s_n = (X_n^1, \dots, X_n^m)$, $v^k(n) = \text{card}(F_n^{-1}(M_k))$ and since u is the *admissible policy*. $\forall k = 1, \dots, m$ functions $F_n^{-1}(M_k)$ satisfy the constraint (9).

6.2 Closed Loop Control

In the *closed loop* algorithm the choice of the first decision in the time epoch τ_μ is analogous to the *open loop* algorithm. The choice of the consecutive decisions (concerning the task subsets allocation) is based on the forecasted average states of the whole workload for each machine in H ($\overline{\Pi}^k(n)$ distributions) generated in each time epoch τ_n , $n = \mu + 1, \dots, \mu + Z - 1$. In other words, the choice of action is based on the forecasted computation time of elementary task generated in each time epoch τ_n , $n = \mu + 1, \dots, \mu + Z - 1$.

Algorithm 2

1. $V(n) = V_D$;
 FOR(each $M_j \in H$)
 Let $\lambda_j(\mu) = \frac{1}{E_{\beta_\mu}(\overline{T}_t^{M_j}(\mu))}$;
 FOR(each $M_j \in H$) /* computation is based on $\Pi^j(\mu)$ */
 Let $F_\mu^{-1}(M_j) \subset V(\mu)$ such that $\text{card}(F_\mu^{-1}(M_j)) = \lfloor \Delta_\mu \cdot \lambda_j(\mu) \rfloor$;
 /** start execution in the Δ_μ */
 $n = \mu + 1$;
 $V(n+1) = V(n) \setminus \bigcup_j F_n^{-1}(M_j)$;
- 2. FOR(each $M_j \in H$) /* computation is based on $\overline{\Pi}^j(n)$ */
 Let $\lambda_j(n) = \frac{1}{E_{\beta_\mu}(\overline{T}_t^{M_j}(n))}$;
 $C = \sum_{M_j \in H} \lfloor \Delta_n \cdot \lambda_j \rfloor$;
- 3. IF($C < \text{card}(V)$)
 { FOR(each $M_j \in H$)
 Let $F_n^{-1}(M_j) \subset V(n)$ such that $\text{card}(F_n^{-1}(M_j)) = \lfloor \Delta_n \cdot \lambda_j(n) \rfloor$;
 /** start execution in the Δ_n */
 $V(n+1) = V(n) \setminus \bigcup_j F_n^{-1}(M_j)$;
 $n = n + 1$;
 GO TO 2;
 }
 ELSE /* last step */
 { $A = \sum_{M_j} \lambda_j(n)$;
 FOR(each $M_j \in H$)
 $\varrho_j = \frac{\lambda_j(n)}{A}$;
 FOR(each $M_j \in H$)
 Let $W_j = F_n^{-1}(M_j)$ such that the set $\left\{ \left| \frac{\text{card}(F_n^{-1}(M_j))}{\text{card}(V(n))} - \varrho_j \right| \right\}$

is minimal over $\{W_j\}_{M_j \in H}$ and $\bigcup_j W_j = V(n)$;

The expected time of a single task execution on the machine M_k in each next time step $n = \mu, \mu + 1 \dots$ is as follows ([8]):

$$E_{\beta_\mu}(\bar{T}_t^k(n)) = E_{\beta_\mu}(\varphi_2^k \circ \varphi_1^k(\bar{X}_n^k)) = \sum_{j=0}^K (\bar{\Pi}_j^k(n) \cdot (1 + \eta_k(j)) \cdot T_t^k) \quad (24)$$

but the expected cost for one step $C_n(\beta_\mu, s_n, u)$ of the realized policy $u = \{a_n\}_{n=\mu+1, \mu+2, \dots} = \{F_n\}_{n=\mu+1, \mu+2, \dots}$ is as follows ([8]):

$$C_n(\beta_\mu, s_n, a_n) = \max_{a_n^k \in A_{s_n^k}^k} \left\{ v(a_n^k) \cdot \sum_{j=0}^K (\bar{\Pi}_j^k(n) \cdot (1 + \eta_k(j)) \cdot T_t^k) \right\} \quad (25)$$

where $s_n = (s_n^1, \dots, s_n^m)$, $a_n = (a_n^1, \dots, a_n^m)$, and because u is the *admissible policy* so. $\forall n \forall k$ a_n^k it has to satisfy constraint (9).

But the cost of the first time step i.e. for $n = \mu$ is as follows:

$$E_{\beta_\mu}(\bar{T}_t^k(n)) = E_{\beta_\mu}(\varphi_2^k \circ \varphi_1^k(\bar{X}_n^k)) = \sum_{j=0}^K (\bar{\Pi}_j^k(n) \cdot (1 + \eta_k(j)) \cdot T_t^k) \quad (26)$$

7 Numerical Tests

The numerical tests concerns the problem of determining the piezometric height distribution during the filtration of water through the cohesive porous medium for a small filtration velocity. The topological decomposition (*SBS computation*) method was utilized to solve it. The main stages of this complex method for the stationary problems are presented on the Fig.1 ([8]). For the nonstationary problems the presented stages are repeated in each time step (nonlinear solver).

No. of tasks	single distribution	multiple task policy	work-greedy policy
100	1.5163	1.9428	2.6339
200	1.5694	1.9515	2.6105

Table 1. The *Speedup* values for the deterministic policies which are realized under medium additional load

The two most time complex parts of SBS computation are the *matrix formulation* and *non linear algebraic solver*. Therefore, the models and algorithms presented in this paper

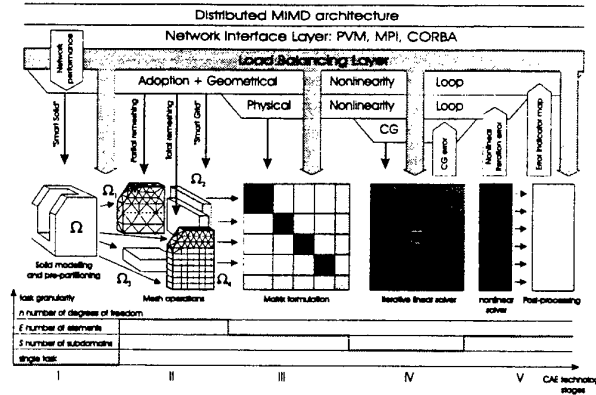


Figure1: Computational technology for the CFG problems

was utilized to the distributed computations of these two stages. Detail description of the mentioned CFG method and the engineering problem under consideration, and the solving method is derived in [5, 7, 8].

The used network H consists of SPARC 4 (as master), two SPARC ELC, SUN 490 and CONVEX 3200 as vector node. During the process of identification of Markov chain we assume that it is periodic with period equal to 24 hours and time intervals $\Delta_n = 1$ hour (see [7]). We obtain speedup values for deterministic policies applied under medium background load for the above numerical application consisting of 100 and 200 tasks (see Table 1, and see [7] for more details of this experiment). These results show various effectiveness of different deterministic policies for fine grain application.

machine	$\theta(0)$	$\theta(1)$	$\theta(2)$
SUN 490	0	1.1	1.9
SPARC ELC	0	0.8	1.2
CONVEX 3200	0	8.0	9.8

Table 2. The sets of thresholds for machines SUN490, SUN SPARC ELC and CONVEX 3200

In case of Markov background load model we assume the uniform space of load states $S = \{0, 1, 2\}$ for each machine in H (i.e. $K = 2$ and $\forall M_j \in H S_j = S$). The state $s = 0$ is the state without background load and $s = 2$ is the state with the greatest background load. The average values for the sets of thresholds Θ_j for machines SUN490, SPARC ELC and CONVEX 3200 are presented in the Table 2. They are determined on the basis of a great number of tests performed for the same pattern application application and defined so that the sets of thresholds Γ_j for slowing down parameters were identical for each machine in H . So, $\forall M_j \in H$

$$\varphi_j(s) = \begin{cases} 0 & \text{if } s = 0 \\ 0.4 & \text{if } s = 1 \\ 0.8 & \text{if } s = 2 \end{cases} \quad (27)$$

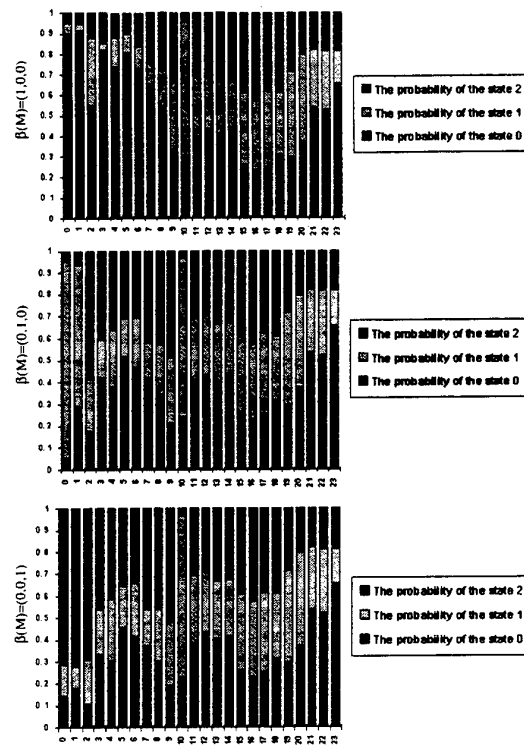


Figure2: The probability distributions for three initial distributions

For example, if the value of the background load ($load_j$) measured by the *state agent* for the machine M_j belongs to the period $[\theta_j(1), \theta_j(2))$ or equivalently (φ_j^1 is a bijection) $\varphi_j^1(load_j) \in [\eta_j(1), \eta_j(2))$ then the state of the background load is equal to $\Psi_j(\psi_j(load_j)) = 1$. Then having the time of single task execution without background load $T_i^j = 2.1$ sec the execution time with the additional background load is equal to $\bar{T}_i^j = 1.4 \cdot 2.1 = 2.94$ sec.

work-greedy policy	stationary policy	closed-loop policy
11931.5 [sec]	12646.2 [sec]	12217.8 [sec]

Table 3. The execution times for the fine grain application consisting of 10000 identical tasks.

The Fig.2 shows the probability distributions for the case of three possible initial distributions for LAN server SUN 490. We can see that only distributions for the first hours strongly depend on the initial distributions and further distributions do not depend on it. It means that the process is ergodic. Some other results involving Markov chain estimation were presented in [7, 8].

Several tests were performed in order to show the influence of the process granularity on the control results (see [7,8]). The comparison of the deterministic and proposed *closed loop*

stochastic control algorithm for the computation of matrix coefficients (*matrix formulation* stage - Fig.1) which is fine grain part of CFG computations is presented in the Table 3. The same algorithms were utilized to the control of the coarse grain computation. Such granularity for the matrix coefficients computations were obtained through the division of the whole set of elementary tasks into the several subsets with the different cardinal numbers. The table 4 presents the cardinal numbers of subsets - the new coarse grain tasks, the number of them, and the results of computations.

Number of elementary tasks in the subset	80	40	14	2	1
No of subsets i.e. coarse grain tasks	60	80	120	80	160
<i>work-greedy</i> policy	stationary policy		<i>closed loop</i> policy		
13616.3 [sec]	15362.9 [sec]		12760.3 [sec]		

Table 4. The execution times for the coarse grain application

The next group of tests were performed in order to compare the stationary policy ([8. 7]) and proposed algorithm of the *open loop* control applied to the second stage of the CFG technology i.e. *iterative linear algebraic solver* (Fig.1). In this case there were 16 similar tasks (one task needs about 3.5 MB of memory) - each one was involved with the similar rectangular subdomain (the generated mesh consisted of 100x160 nodes and about 32000 triangular elements) and the horizon $Z = 1$ (the control for one time step).

	static control, 16 coarse grain tasks				
times of					
5 tests	4304 sec	2764 sec	4638 sec	3693 sec	3952 sec

Table 5. The execution times of SBS-PCG application consisting of 16 coarse grain tasks for static control

In case of the static control which is based on the state of each machine of the network in the starting time epoch we obtained the results presented in the Table ???. The next tests involved *open loop* control were performed for the case of the Markov forecast. The results for the background load increase and decrease forecast were presented in the Table 6.

8 Conclusions

1. Stochastic policies are better suited to the coarse grain problem of tasks distribution e.g. PSG-SBS solution of large-scale linear systems then the deterministic ones.
2. The *closed loop control* usually is better then the *open loop* one in case of computation on the horizon consisting of more then one time period because it provides the

	the execution times for Markov control, 16 coarse grain tasks		
the background load increase forecast	2438 sek	2588 sek	2306 sek
the background load decrease forecast	2940 sek	2765 sek	2852 sek

Table 6. The execution times for the SBS-PCG application consisting of 16 coarse grain tasks for Markov control

- better estimation of computational power of each $M \in H$ for each time period during computation. In other words, the probability that *closed loop control* of the distributed computations gives better execution times than *open loop control*, is greater then zero.
3. The presented models of network workload and stochastic policies of task distribution can be utilized not only to CAE distributed programs managing as well as to another scalable large distributed applications.

REFERENCES

- [1] Donaldson, F. Berman, R. Paturi, *Program Speedup in Heterogeneous Computing Network*, Journal of Parallel and Distributed Computing 21, 316-332 (1994).
- [2] S. Manoharan, N.P. Topham, *An assessment of assignment schemes for dependency graphs*, Parallel Computing 21 (1995) 85-107.
- [3] Puterman M.L., *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, John Wiley & Sons, Inc, 1994.
- [4] E.Altman, *Constrained Markov Decision Process*, INRIA, Centre Sophia Antipolis. France, Raport No.2574, 1995
- [5] R. Schaefer, S. Sedziwy, *Filtration in cohesive soils: modelling and solving in*: Proc. of the Conf.:Finite Elements in Fluids, New Trends and Appl.,Barcelona 1993, Vol. II., 887 - 891
- [6] H. Kuszner, *Intrudaction to Sochastic Control*, Holt, Reinhart and Winston. Inc., New York Copyright 1971.
- [7] Onderka Z., Schaefer R., *Markov Chain Based Management of Large Scale Distributed Computations of Earthen Dam Leakages*, the 2nd Intern. Meeting on Vector and Parallel Processing (Systems and Applications) VECPAR'96, September 1996, Porto. Portugal - Lecture Notes in Computer Science, Palma Jose M.L.M., Dongarra J., No.1215. 1997, pp. 49-64, Springer-Verlag.
- [8] Onderka Z. *Stochastic Control of the Distributed Scalable Applications. Application in the CAE Technology*, Doctoral Thesis, Technical University, Department of Computer Science, Krakow, Poland, 1997 (english version in preparation).
- [9] Onderka Z. *Markov Management of the CAE Distributed Applications*, Proceedings of the XIII Intern. Conf. on Computer Methods in Mechanics, Poznan, Poland, May 1997, Garstecki A., Rakowski J. (Eds) pp 1009-1013 vol3.

Direct Linear Solvers for Vector and Parallel Computers*

Friedrich Grund

Weierstrass Institute for Applied Analysis and Stochastics
Mohrenstrasse 39, 10117 Berlin, Germany
grund@wias-berlin.de
<http://www.wias-berlin.de/~grund>

Abstract. We consider direct methods for the numerical solution of linear systems with unsymmetric sparse matrices. Different strategies for the determination of the pivots are studied. For solving several linear systems with the same pattern structure we generate a pseudo code, that can be interpreted repeatedly to compute the solutions of these systems. The pseudo code can be advantageously adapted to vector and parallel computers. For that we have to find out the instructions of the pseudo code which are independent of each other. Based on this information, one can determine vector instructions for the pseudo code operations (vectorization) or spread the operations among different processors (parallelization). The methods are successfully used on vector and parallel computers for the circuit simulation of VLSI circuits as well as for the dynamic process simulation of complex chemical production plants.

1 Introduction

For solving systems of linear equations

$$Ax = b, \quad A \in \mathbb{R}^{n \times n}, \quad x, b \in \mathbb{R}^n \quad (1)$$

with non singular, unsymmetric and sparse matrices A , we use the Gaussian elimination method. Only the nonzero elements of the matrices are stored for computation. In general, we need to establish a suitable control for the numerical stability and for the fill-in of the Gaussian elimination method.

For the time domain simulation in many industrial applications structural properties are used for a modular modeling. Thus electronic circuits usually consist of identical subcircuits as inverter chains or adders. Analogously, complex chemical plants consist of process units as pumps, reboilers or trays of distillation columns. A mathematical model is assigned to each subcircuit or unit and they are coupled. This approach leads to initial value problems for large systems of differential-algebraic equations. For solving such problems we use backward differentiation formulas and the resulting systems of nonlinear equations are

* This work was supported by the Federal Ministry of Education, Science, Research and Technology, Bonn, Germany under grants GA7FVB-3.0M370 and GR7FV1.

solved with Newton methods. The Jacobi matrices are sparse and maintain their sparsity structure during the integration over many time steps. In general, the Gaussian elimination method can be used with the same ordering of the pivots for these steps. A pseudo code is generated to perform the factorizations of the matrices and the solving of the systems with triangular matrices efficiently. This code contains only the required operations for the factorization and for solving the triangular systems. It is defined independently of a computer and can be adapted to vector and parallel computers.

The solver has been proven successfully for the dynamic process simulation of large real life chemical production plants and for the electric circuit simulation as well. Computing times for complete dynamic simulation runs of industrial applications are given. For different linear systems with matrices arising from scientific and technical problems the computing times for several linear solvers are compared.

2 The method

The Gaussian elimination method

$$PAQ = LU, \quad (2)$$

$$Ly = Pb, \quad UQ^{-1}x = y \quad (3)$$

is used for solving the linear systems (1). The nonzero elements of the matrix A are stored in compressed sparse row format, also known as sparse row wise format. L is a lower triangular and U an upper triangular matrix. The row permutation matrix P is used to provide numerical stability and the column permutation matrix Q is used to control sparsity. In the following, we consider two cases for the determination of the matrices P and Q .

In the first case, we determine in each elimination step a permutation in the matrix Q . For this, we search the first column with a minimal number of nonzero elements in the matrix to be eliminated. This column becomes the pivot column [6] and the columns are reordered (dynamic ordering). For keeping the method numerically stable at stage k of the elimination, the pivot $a_{i,j}$ is selected among those candidates satisfying the numerical threshold criterion

$$|a_{i,j}| \geq \beta \max_{l \geq k} |a_{l,j}|$$

with a given threshold parameter $\beta \in (0, 1]$. This process is called partial pivoting. In our applications we usually choose $\beta = 0.01$ or $\beta = 0.001$.

In the second case, we determine in a first step the permutation matrix Q by minimum degree ordering of $A^T A$ or of $A^T + A$, using the algorithm from SuperLU [9]. Then the columns are reordered and in a separate step the permutation matrix P is determined by using partial pivoting.

3 Pseudo code

As mentioned above, it is possible to use the Gaussian elimination method with the same pivot ordering to solve several linear systems with the same pattern structure of the coefficient matrix. To do this, we generate a pseudo code to perform the factorization of the matrix as well as to solve the triangular systems (forward and back substitution).

For the generation of the pseudo code, the factorization of the Gaussian elimination method is used as shown in Fig. 1.

```

for i = 2, n do
  ai-1,i-1 = 1/ai-1,i-1
  for j = i, n do
    aj,i-1 = (aj,i-1 - ∑k=1i-2 aj,kak,i-1)ai-1,i-1
  enddo
  for j = i, n do
    ai,j = ai,j - ∑k=1i-1 ai,kak,j
  enddo
enddo
an,n = 1/an,n.

```

Fig. 1. Gaussian elimination method

The algorithm needs n divisions. Six different types of pseudo code instructions are sufficient for the factorization of the matrix, four instructions for the computation of the elements of the upper triangular matrix and two of the lower triangular matrix. For computing the elements of the upper triangular matrix one has to distinguish between the cases that the element is a pivot or not and that it exists or that it is generated by fill-in. For the determination of the elements of the lower triangular matrix one has only to distinguish that the element exists or that it is generated by fill-in.

Let l , with $1 \leq l \leq 6$, denote the type of the pseudo code instruction, n the number of elements of the scalar product and $k, m, i_\kappa, j_\kappa, \kappa = 1, 2, \dots, n$ the indices of matrix elements. Then, the instruction of the pseudo code to compute an element of the lower triangular matrix

$$a(k) = \left(a(k) - \sum_{\kappa=1}^n a(i_\kappa) a(j_\kappa) \right) a(m)$$

is coded in the following form

l	n	i ₁	j ₁	...	i _n	j _n	k	m
---	---	----------------	----------------	-----	----------------	----------------	---	---

The integer numbers $l, n, i_\kappa, j_\kappa, k$ and m are stored in integer array elements. For l and n only one array element is used.

The structure of the other pseudo code instructions is analogous.

Let μ denote the number of multiplications and divisions for the factorization of the matrix and ν the number of nonzero elements of the upper and lower triangular matrices. Then one can estimate the number of integer array elements that are necessary to store the pseudo code with

$$\gamma(\mu + \nu).$$

At this $\gamma \approx 2.2$ was found to be sufficient for large systems with more than thousand equations while one has to choose $\gamma \approx 4$ for smaller systems.

4 Vectorization and parallelization

The pseudo code instructions are used for the vectorization and the parallelization as well. For the factorization in (2) and for solving the triangular systems in (3), elements have to be found that can be computed independently of each other.

In the case of the factorization, a matrix

$$M = (m_{i,j}), \quad m_{i,j} \in \mathbb{N} \cup \{0, 1, 2, \dots, n^2\}$$

is assigned to the matrix

$$LU = PAQ,$$

where $m_{i,j}$ denotes the level of independence.

In the case of solving the triangular systems, vectors

$$p = (p_i) \quad \text{and} \quad q = (q_i), \quad p_i, q_i \in \{0, 1, \dots, n\}$$

are assigned analogously to the vectors x and y from

$$Ly = Pb \quad \text{and} \quad UQ^{-1}x = y.$$

Here the levels of independence are denoted by p_i and q_i .

The elements with the assigned level zero do not need any operations. Now, all elements with the same level in the factorized matrix (2) as well as in the vectors x and y from (3) can be computed independently. First all elements with level one are computed, then all elements with level two and so on.

The levels of independence for the matrix elements in (2) and for the vector elements in (3) can be computed with the algorithm of Yamamoto and Takahashi [11]. The algorithm for the determination of the levels of independence $m_{i,j}$ is shown in Fig. 2. The corresponding algorithm for the determination of the elements of the vectors p and q is analogous to it.

```

M = 0
for i = 1, n - 1 do
  for all {j : aj,i ≠ 0 & j > i} do
    mj,i = 1 + max(mj,i, mi,i)
    for all {k : ai,k ≠ 0 & k > i} do
      mj,k = 1 + max(mj,k, mj,i, mi,k)
    enddo
  enddo
enddo.

```

Fig. 2. Algorithm of Yamamoto and Takahashi

For a vector computer, we have to find vector instructions at the different levels of independence [2, 7]. Let $a(i)$ denote the nonzero elements in LU . The vector instructions, shown in Fig. 3, have been proven to be successful in the case of factorization. The difficulty is that the array elements are addressed indirectly. But adequate vector instructions exist for many vector computers. The Cray vector computers, for example, have explicit calls to gather/scatter routines for the indirect addressing.

$$s = \sum_{\kappa} a(i_{\kappa}) * a(j_{\kappa})$$

$$a(i_k) = 1/a(i_k)$$

$$a(i_k) = a(i_k) * a(i_l)$$

$$a(i_k) = (a(i_l) * a(i_m) + a(i_p) * a(i_q)) * a(i_k)$$

Fig. 3. Types of vector instructions for factorization

For parallelization, it needs to distinguish between parallel computers with shared memory and with distributed memory.

In the case of parallel computers with shared memory and p processors, we assign the pseudo code for each level of independence in parts of approximately same size to the processors. After the processors have executed their part of the pseudo code instructions of a level concurrently, a synchronization among the processors is needed. Then the execution of the next level can be started. If the processors are vector processors then this property is also used. The moderate parallel computer Cray J90 with a maximum number of 32 processors is an example for such a computer.

In the case of parallel computers with distributed memory and q processors, the pseudo code for each level of independence is again partitioned into q parts

of approximately same size. But in this case, the parts of the pseudo code are moved to the memory of each individual processor. The transfer of parts of the code to the memories of the individual processors is done only once. A synchronization is carried out analogous to the shared memory case. The partitioning and the storage of the matrix as well as of the vectors is implemented in the following way. For small problems the elements of the matrix, right hand side and solution vector are located in the memory of one processor, while for large problems, they have to be distributed over the memories of several processors. We assume that the data communication between the processors for the exchange of data concerning elements of the matrix, right hand side and solution vector is supported by the operating system. The massive parallel computers Cray T3D and T3E are examples for such computers.

Now, we consider a small example to illustrate our approach. For a matrix

$$A = \begin{pmatrix} 9 & 2 & 1 \\ 1 & 3 & 5 \\ & 2 & 4 \\ 1 & 7 & 8 \\ & 5 & 7 & 9 \end{pmatrix} \quad (4)$$

the determination the permutation matrices P and Q gives

$$PAQ = \begin{pmatrix} 2 & 4 & & \\ 5 & 7 & & 9 \\ & 2 & 9 & 1 \\ & & 1 & 7 & 8 \\ & & & 1 & 3 & 5 \end{pmatrix}. \quad (5)$$

The nonzero elements of the matrix A are stored in sparse row format in the vector a . Let \boxed{i} denote the index of the i -th element in the vector a , then the elements of the matrix PAQ are stored in the following way

$$\begin{pmatrix} \boxed{7} & \boxed{8} & & & & \\ \boxed{12} & \boxed{13} & & & & \boxed{14} \\ & \boxed{2} & \boxed{1} & & & \boxed{3} \\ & & \boxed{9} & \boxed{10} & \boxed{11} & \\ & & \boxed{4} & \boxed{5} & \boxed{6} & \end{pmatrix}. \quad (6)$$

The matrix M assigned to the matrix PAQ is found to be

$$M = \begin{pmatrix} 0 & 0 & & & & \\ 1 & 2 & & & & 0 \\ & 3 & 0 & & & 4 \\ & & 1 & 0 & 5 & \\ & & & 1 & 1 & 6 \end{pmatrix}. \quad (7)$$

From (7), we can see, that six independent levels exist for the factorization.

The instructions for the factorization of the matrix A resulting from (4) - (7) are shown in Table 1.

Table 1. Instructions for the factorization

Level	Instructions
1	$a(12) = a(12)/a(7)$
	$a(9) = a(9)/a(1)$
	$a(4) = a(4)/a(1)$
	$a(5) = a(5)/a(10)$
2	$a(13) = a(13) - a(12) * a(8)$
3	$a(2) = a(2)/a(13)$
4	$a(3) = a(3) - a(2) * a(14)$
5	$a(11) = a(11) - a(5) * a(3)$
6	$a(6) = a(6) - a(4) * a(3) - a(5) * a(11)$

Now, we consider, for example, the instructions of level one in Table 1 only. One vector instruction of the length four can be generated (see Fig.3) on a vector computer.

On a parallel computer with distributed memory and two processors, the allocation of the instructions of level one to the processors is shown in Table 2. The transfer of the instructions to the local memory of the processors is done during the analyse step of the algorithm. The data transfer is carried out by the operating system.

Table 2. Allocation of instructions to processors

	processor one	processor two
computation of	$a(12), a(9)$	$a(4), a(5)$
	synchronization	

On a parallel computer with shared memory the approach is analogous. The processors have to be synchronized after the execution of the instructions of each level.

From our experiments with many different matrices arising from the process simulation of chemical plants and the circuit simulation respectively, it was found that the number of levels of independence is small. The number of instructions in the first two levels is very large, in the next four to six levels it is large and finally it becomes smaller and smaller.

5 Numerical results

The developed numerical methods are realized in the program package GSPAR. GSPAR is implemented on workstations (Digital AlphaStation, IBM RS/6000, SGI, Sun UltraSparc 1 and 2), vector computers (Cray J90, C90), parallel computers with shared memory (Cray J90, C90, SGI Origin2000, Digital AlphaServer) and parallel computers with distributed memory (Cray T3D).

The considered systems of linear equations result from real life problems in the dynamic process simulation of chemical plants, in the electric circuit simulation and in the account of capital links (political sciences)¹. The $n \times n$ matrices A with $|A|$ nonzero elements are described in Table 3.

Table 3. Test matrices

name	discipline	n	A
bayer01	chemical	57 735	277 774
b_dyn	engineering	1 089	4 264
bayer02		13 935	63 679
bayer03		6 747	56 196
bayer04		20 545	159 082
bayer05		3 268	27 836
bayer06		3 008	27 576
bayer09		3 083	21 216
bayer10		13 436	94 926
advice3388	circuit	33 88	40 545
advice3776	simulation	3 776	27 590
cod2655_tr		2 655	24 925
meg1		2 904	58 142
meg4		5 960	46 842
rlxADC_dc		5 355	24 775
rlxADC_tr		5 355	32 251
zy3315		3 315	15 985
poli	account of	4 008	8 188
poli_large	capital links	15 575	33 074

In Table 4 results for the matrices in Table 3 are shown using the method GSPAR on a DEC AlphaServer with an alpha EV5.6 (21164A) processor. Here, $\# op LU$ is the number of operations (only multiplications and divisions) and $fill-in$ is the number of fill-ins during the factorization. The cpu time (in seconds)

¹ Some matrices, which are given in Harwell-Boeing format and interesting details about the matrices, can be found in Tim Davis, University of Florida Sparse Matrix Collection, <http://www.cise.ufl.edu/~davis/sparse/>

for the first factorization, presented in *strat*, includes the times for the analysis as well as for the numerical factorization. The cpu time for the generation of the pseudo code is given in *code*. At the one hand, a dynamic ordering of the columns can be applied during the pivoting. At the other hand, a minimum degree ordering of $A^T A$ (upper index *) or of $A^T + A$ (upper index +) can be used before the partial pivoting.

Table 4. GSPAR first factorization and generation pseudo code

name	dynamic ordering					minimum degree ordering				
	# op	LU	fill-in	strat.	code	# op	LU	fill-in	strat.	code
bayer01	10 032 621	643 898	35.18	12.72		13 860 173	812 505	5.75	9.95 *	
b_dyn	15 902	2 909	0.02	0		21 556	8 231	0.02	0.02 *	
bayer02	2 095 207	134 546	2.28	1.30		2 030 130	165 357	1.03	2.20 *	
bayer03	1 000 325	64 130	0.68	0.47		625 272	53 991	0.25	0.35 *	
bayer04	5 954 718	268 006	5.33	3.93		6 340 579	290 021	1.95	2.77 *	
bayer05	119 740	11 024	0.15	0.03		474 273	33 797	0.18	0.17 *	
bayer06	3 042 620	73 773	0.85	1.00		5 008 097	129 278	1.42	1.52 *	
bayer09	364 731	23 145	0.18	0.15		287 947	22 022	0.12	0.12 *	
bayer10	5 992 500	227 675	3.05	2.55		3 953 687	203 633	1.28	1.40 *	
advice3388	310 348	9 297	0.38	0.65		396 965	9 818	0.75	0.95 +	
advice3776	355 465	25 656	0.35	0.75		382 224	26 074	0.62	0.98 +	
cod2655_tr	3 331 105	113 640	0.90	1.00		4 839 771	144 875	1.50	1.40 +	
meg1	796 797	40 436	0.32	0.40		1 245 847	59 558	0.48	0.78 +	
meg4	420 799	38 784	0.68	0.62		376 324	35 008	0.30	0.48 +	
rlxADC_dc	73 612	5 404	0.38	0.13		63 227	2 906	0.08	0.08 +	
rlxADC_tr	988 759	47 366	0.85	1.13		1 049 623	48 888	0.72	1.13 +	
zy3315	47 326	8 218	0.12	0.03		49 263	8 202	0.03	0.02 +	
poli	4 620	206	0.15	0		6 094	41	0.02	0 *	
poli_large	43 310	10 318	2.38	0.25		34 115	588	0.08	0.03 +	

The results in Table 4 show the following characteristics. For linear systems arising from the process simulation of chemical plants, the analyse step with the minimum degree ordering is in most cases, particularly for large systems, faster than with the dynamic ordering, but the fill-in and the number of operations for the factorization are larger. On the other hand, for systems arising from the circuit simulation the factorization with the dynamic ordering is in most cases faster than the minimum degree ordering. The factorization with the minimum degree ordering of $A^T A$ is favourable for systems arising from chemical process simulation, while using an ordering of $A^T + A$ is recommendable for systems arising from the circuit simulation. The opposite cases of the minimum degree ordering are unfavourable because the number of operations and the number of fill-ins is very large.

In Table 5, cpu times (in seconds) for the second factorization are shown for the linear solvers UMFPACK [4], SuperLU with minimum degree ordering of $A^T A$ (upper index *) or of $A^T + A$ (upper index +) [5], Sparse [8] and GSPAR with dynamical column ordering, using a DEC AlphaStation with an alpha EV4.5 (21064) processor. In many applications, mainly in the numerical simulation of physical and chemical problems, the analysis step including ordering and first factorization is performed only a few times, but the second factorization is performed often. Therefore the cpu time for the second factorization is essential for the overall simulation time.

Table 5. Cpu times for second factorization

name	UMFPACK	SuperLU	Sparse	GSPAR
bayer01	5.02	6.70 *	7.78	3.20
b_dyn	0.05	0.05 *	0.07	0.00
bayer02	1.13	1.47 *	10.433	0.55
bayer03	0.72	0.70 *	17.467	0.27
bayer04	3.37	2.77 *	187.88	1.70
bayer05	0.13	0.75 *	0.08	0.05
bayer06	0.83	0.90 *	54.33	0.82
bayer09	0.23	0.23 *	3.57	0.10
bayer10	1.60	1.57 *	379.75	1.65
advice3388	0.25	0.28 +	0.15	0.10
advice3776	0.30	0.42 +	0.20	0.10
cod2655_tr	0.30	0.55 +	0.27	0.10
meg1	0.58	1.43 +	13.95	0.22
meg4	0.37	0.75 +	0.25	0.13
rlxADC_dc	0.15	0.18 +	0.04	0.03
rlxADC_tr	0.40	0.90 +	0.72	0.30
zy3315	0.15	0.18 +	0.03	0.02
poli	0.03	0.07 +	0.00	0.00
poli_large	0.13	0.27 +	0.04	0.03

GSPAR achieves a fast second factorization for all linear systems in Table 5. For linear systems with a large number of equations GSPAR is at least two times faster than UMFPACK, SuperLU and Sparse respectively.

The cpu times for solving the triangular matrices are one order of magnitude smaller than the cpu times for the factorization. The proportions between the different solvers are comparable to the results in Table 5.

The vector version of GSPAR has been compared with the frontal method FAMP [12] on a vector computer Cray Y-MP8E using one processor. The used version of FAMP is the routine from the commercial chemical process simulator

SPEEDUP² [1]. The cpu times (in seconds) for the second factorization are shown in Table 6.

Table 6. Cpu times for second factorization

name	FAMP	GSPAR
b_dyn	0.034	0.011
bayer09	0.162	0.082
bayer03	0.404	0.221
bayer02	0.683	0.421
bayer10	1.290	0.738
bayer04	2.209	0.983

GSPAR is at least two times faster than FAMP for these examples. The proportions for solving the triangular systems are again the same.

For two large examples the number of levels of independence are given in Table 7, using GSPAR with two different ordering for pivoting. The algorithm for lower triangular systems is called forward substitution and the analogous algorithm for upper triangular systems is called back substitution.

Table 7. Number of levels of independence

example		dynamical ordering	minimum degree ordering
bayer01	factorization	3 077	3 688
	forward sub.	1 357	1 562
	back substit.	1 728	2 476
bayer04	factorization	876	820
	forward sub.	399	338
	back substit.	556	495

In Table 8, wall-clock times (in seconds) are shown for the second factorization, using GSPAR with different pivoting on a DEC AlphaServer with four alpha EV5.6 (21164A) processors. The parallelization technique is based on OpenMP [10]. The wall-clock times have been determined with the system routine *gettimeofday*.

In Table 9, the cpu times (in seconds) on a Cray T3D are given for the second factorization, using GSPAR with dynamic ordering for pivoting. The linear

² Used under licence 95122131717 for free academic use from Aspen Technology, Cambridge, MA, USA; Release 5.5-5

Table 8. Wall-clock times for second factorization

processors	dynamical ordering		minimum degree ordering	
	bayer01	bayer04	bayer01	bayer04
1	0.71	0.39	1.08	0.43
2	0.54	0.27	0.75	0.29
3	0.45	0.23	0.63	0.25
4	0.49	0.24	0.70	0.30

systems can not be solved with less than four or sixteen processors respectively, because the processors of the T3D have not enough local memory for the storage of the pseudo code in this cases. The speedup factors are set equal to one for four or sixteen processors respectively.

Table 9. Cpu times for second factorization on Cray T3D

example	processors	cpu time	speedup factor
bayer04	4	1.59	1.00
	8	0.99	1.60
	16	0.60	2.65
	32	0.37	4.30
	64	0.24	6.63
bayer01	16	2.36	1.00
	32	1.45	1.63
	64	0.95	2.47

6 Applications

Problems of the dynamic process simulation of chemical plants can be modeled by initial value problems for systems of differential-algebraic equations. The numerical solution of these systems [3] involves the solution of large scale systems of nonlinear equations, which can be solved with modified Newton methods. The Newton corrections are found by solving large unsymmetric sparse systems of linear equations. The overall computing time of the simulation problems is often dominated by the time needed to solve the linear systems. In industrial applications, the solution of sparse linear systems requires often more than 70 % of the total simulation time. Thus a reduction of the linear system solution time usually results into a significant reduction of the overall simulation time [13].

Table 10 shows three large scale industrial problems of the Bayer AG Leverkusen. The number of differential-algebraic equations as well as an estimate for the condition number of the matrices of the linear systems are given. The condition numbers are very large, what is typical for industrial applications in this field.

Table 10. Large scale industrial problems

name	chemical plants	equations	condition numbers
bayer04	nitration plant	3 268	2.95E+26, 1.4E+27
bayer10	distillation column	13 436	1.4E+15
bayer01	five coupled distillation columns	57 735	6.0E+18 6.96E+18

The problems have been solved on a vector computer Cray C90 using the chemical process simulator SPEEDUP [1]. In SPEEDUP the vector versions of the linear solvers FAMP and GSPAR have been used alternatively. The cpu time (in seconds) for complete dynamic simulation runs are shown in Table 11.

Table 11. Cpu time for complete dynamic simulation

name	FAMP	GSPAR	in %
bayer04	451.7	283.7	62.8
bayer10	380.9	254.7	66.9

For the large plant bayer01 benchmark tests have been performed on a dedicated computer Cray J90, using the simulator SPEEDUP with the solvers FAMP and GSPAR alternatively. The results are given in Table 12.

Table 12. Bench mark tests

time	FAMP	GSPAR	in %
cpu time	6 066.4	5 565.8	91.7
wall-clock time	6 697.9	5 797.1	86.5

The simulation of plant bayer01 has been performed also on a vector computer Cray C90 connected with a parallel computer Cray T3D, using SPEEDUP

and the parallel version of GSPAR. Here, the linear systems have been solved on the parallel computer while the other parts of the algorithms of SPEEDUP have been performed on the vector computer. GSPAR needs 1 440.5 seconds cpu time on a T3D with 64 used processors. When executed on the Cray C90 only, 2 490 seconds are needed for the total simulation.

Acknowledgments. The author thanks his coworkers J. Borchardt and D. Horn for useful discussions. The valuable assistance and the technical support from the Bayer AG Leverkusen, the Cray Research Munich and Aspen Technology, Inc., Cambridge, MA, USA are gratefully acknowledged.

References

1. AspenTech: SPEEDUP, User Manual, Library Manual. Aspen Technology, Inc., Cambridge, Massachusetts, USA (1995)
2. Borchardt, J., Grund, F., Horn, D.: Parallelized numerical methods for large systems of differential-algebraic equations in industrial applications. Preprint No. 382, WIAS Berlin (1997). *Surv. Math. Ind.* (to appear)
3. Brenan, K.E., Campbell, S.L., Petzold, L.R.: Numerical solution of initial-value problems in differential-algebraic equations. North-Holland, New York (1997)
4. Davis, T.A., Duff, I.S.: An unsymmetric-pattern multifrontal method for sparse LU factorization. Tech. Report TR-94-038, CIS Dept., Univ. of Florida, Gainesville, FL (1994)
5. Demmel, J.W., Gilbert, J.R. Li, X.S.: SuperLU Users' Guide. Computer Science Division, U.C. Berkeley (1997)
6. Grund, F., Borchardt, J., Horn, D., Michael, T., Sandmann, H.: Differential-algebraic systems in chemical process simulation. In *Scientific Computing in Chemical Engineering*, F. Keil, W. Mackens, H. Voß, J. Werther, eds., Springer-Verlag Berlin Heidelberg (1996) 68-74
7. Grund, F., Michael, T., Brüll, L., Hubbuch, F., Zeller, R., Borchardt, J., Horn, D., Sandmann, H.: Numerische Lösung großer strukturierter DAE-Systeme der chemischen Prozesssimulation. In *Mathematik Schlüsseltechnologie für die Zukunft*, K.-H. Hoffmann, W. Jäger, T. Lohmann, H. Schunk, eds., Springer-Verlag Berlin Heidelberg (1997) 91-103
8. Kundert, K.S., Sangiovanni-Vincentelli, A.: Sparse User's Guide, A Sparse Linear Equation Solver. Dep. of Electr. Engin. and Comp. Sc., U.C. Berkeley (1988)
9. Li, Xiaoye S.: Sparse Gaussian elimination on high performance computers. Technical Reports UCB//CSD-96-919, Computer Science Division, U.C. Berkeley (1996), Ph.D. dissertation
10. OpenMP: A proposed standard API for shared memory programming. White paper, <http://www.openmp.org> (1997)
11. Yamamoto, F., Takahashi, S.: Vectorized LU decomposition algorithms for large-scale nonlinear circuits in the time domain. *IEEE Trans. on Computer-Aided Design CAD-4* (1985) 232-239
12. Zitney, S.E., Stadtherr, M.A.: Frontal algorithms for equation-based chemical process flowsheeting on vector and parallel computers. *Computers chem. Engng.* **17** (1993) 319-338
13. Zitney, S.E., Brüll, L., Lang, L., Zeller, R.: Plantwide dynamic simulation on supercomputers: Modelling a Bayer distillation process. *AIChE Symp. Ser.* **91** (1995) 313-316

The Design of an ODMG Compatible Parallel Object Database Server

Paul Watson

Department of Computing Science, University of Newcastle-upon-Tyne, NE1 7RU, UK
Paul.Watson@newcastle.ac.uk

Abstract. The *Polar* project has the aim of designing a parallel, ODMG compatible object database server. This paper describes the server requirements and investigates issues in designing a system to achieve them. We believe that it is important to build on experience gained in the design and usage of parallel relational database systems over the last ten years, as much is also relevant to parallel object database systems. Therefore we present an overview of the design of parallel relational database servers and investigate how their design choices could be adopted for a parallel object database server. We conclude that while there are many similarities in the requirements and design options for these two types of parallel database servers, there are a number of significant differences, particularly in the areas of object access and method execution.

1 Introduction

The parallel database server has become the "killer app" of parallel computing. The commercial market for these systems is now significantly larger than that for parallel systems running numeric applications, making them mainstream IT system components offered by a number of major computer vendors. They can provide high performance, high availability, and high storage capacity, and it is this combination of attributes which has allowed them to meet the growing requirements of the increasing number of computer system users who need to store and access large amounts of information.

There are a number of reasons to explain the rapid rise of parallel database servers, including:

- they offer higher performance at a better cost-performance ratio than do the previous dominant systems in their market - mainframe computers.
- designers have been able to produce highly available systems by exploiting the natural redundancy of components in parallel systems. High availability is important because many of these systems are used for business-critical applications in which the financial performance of the business is compromised if the data becomes inaccessible.

- the 1990s have seen a process of re-centralisation of computer systems following the trend towards downsizing in the 1980s. The reasons for this include: cost savings (particularly in software and system management), regaining central control over information, improving data integrity by reducing duplication, and increasing access to information. This process has created the demand for powerful information servers.
- there has been a realisation that many organisations can derive and infer valuable information from the data held in their databases. This has led to the use of techniques, such as data-mining, which can place additional load on the database server from which the base data on which they operate must be accessed.
- the growing use of the Internet and intranets as ways of making information available both outside and inside organisations has increased the need for systems which can make large quantities of data available to large numbers of simultaneous users.

The growing importance of parallel database servers is reflected in the design of commercial parallel platforms. Efficient support for parallel database servers is now a key design requirement for the majority of parallel systems.

To date, almost all parallel database servers have been designed to support relational database management systems (RDBMS) [1]. A major factor which has simplified, and so encouraged, the deployment of parallel RDBMS by organisations is their structure. Relational database systems have a client-server architecture in which client applications can only access the server through a single restricted and well defined query interface. To access data, clients must send an SQL (Structured Query Language) query to the server where it is compiled and executed. This architecture allows a serial server which is not able to handle the workload generated by a set of clients to be replaced by a parallel server with higher performance. The client applications are unchanged: they still send the same SQL to the parallel server as they did to the serial server because the exploitation of parallelism is completely internal to the server.

Existing parallel relational database servers exploit two major types of parallelism. Inter-query parallelism is concerned with the simultaneous execution of a set of queries. It is typically used for On-Line Transaction Processing (OLTP) workloads in which the server processes a continuous stream of small transactions generated by a set of clients. Intra-Query parallelism is concerned with exploiting parallelism within the execution of single queries so as to reduce their response time.

Despite the current market dominance of relational database servers, there is a growing belief that relational databases are not ideal for a number of types of applications, and in recent years there has been a growth of interest in object oriented databases which are able to overcome many of the problems inherent in relational systems [2]. In particular, the growth in the use of object oriented programming languages, such as C++ and Java, coupled with the increasing importance of object-based distributed systems has promoted the use of object database management systems (ODBMS) as key system components for object storage and access. A consequence of the interest in object databases has been an attempt to define a

standard specification for all the key ODBMS interfaces - the Object Database Management Group ODMG 2.0 standard [3].

The *Polar* project has the aim of designing and implementing a prototype ODMG compatible parallel object database server. Restricting the scope of the project to this standard allows us to ignore many issues, such as query language design and programming language interfaces, and instead focus directly on methods for exploiting parallelism within the framework imposed by the standard.

In this paper we describe the requirements that the *Polar* server must meet and investigate issues in designing a system to achieve them. Rather than design the system in isolation, we believe that it is important to build, where possible, on the extensive experience gained over the last ten years in the design and usage of parallel relational database systems. However, as we describe in this paper, differences between the object and relational database paradigms result in significant differences in some areas of the design of parallel servers to support them. Those differences in the paradigms which have most impact on the design are:

- objects in an object database can be referenced by a unique identifier, or (indirectly) as members of a collection. In contrast, tables (collections of rows) are the only entities which can be referenced by a client of a relational database (i.e. individual table rows cannot be directly referenced).
- there are two ways to access data held in an object database: through a query language (OQL), and by directly mapping database objects into client application program objects. In a relational database, the query language (SQL) is the only way to access data.
- objects in an object database can have associated user-defined methods which may be called within queries, and by client applications which have mapped database objects into program objects. In a relational database, there is no equivalent of user-defined methods: only a fixed set of operations is provided by SQL.

The structure of the rest of this paper is as follows. In Section 2 we give an overview of the design of parallel relational database servers, based on our experiences in two previous parallel database server projects: EDS [4], and Goldrush [1]. Next, in Section 3 we define our requirements for the *Polar* parallel ODBMS. Some of these are identical to those of parallel RDBMS; others have emerged from experience of the limitations of existing parallel servers; while others are derived from our view of the potential use of parallel ODBMS as components in distributed systems. Based on these requirements, in Section 4, we present an overview of issues in the design of a parallel ODBMS. This allows us to highlight those areas in the design of a parallel object database server where it is possible to adopt solutions based on parallel RDBMS or serial ODBMS, and, in contrast, those areas where new solutions are required. Finally, in Section 5 we draw conclusions from our investigations, and point to further work.

1.1 Related Work

There have been a number of parallel relational database systems described in the literature. Those which have most influenced this paper are the two on which we previously worked: EDS and Goldrush.

The EDS project [4] designed and implemented a complete parallel database server, including hardware, operating system and database. The database itself was basically relational though there were some extensions to provide support for objects.

The Goldrush project within ICL High Performance Systems [1, 5, 6] designed a parallel relational database server product running Parallel Oracle.

There is extensive coverage in the literature of research into the design of serial ODBMS. One of the most complete is the description of the O2 system [7].

Recently, there has been some research into the design of parallel ODBMS. For example, Goblin [8] is a parallel ODBMS. However, unlike the system described in this paper, it is limited to a main-memory database.

Work on object servers such as Shore [9] and Thor [10] is also of relevance as this is a key component of any parallel ODBMS. We will refer to this work at appropriate points in the body of the paper.

2 The Design of Parallel Relational Database Servers

Parallel relational database servers have been designed, implemented and utilised for over ten years, and it is important that this experience is used to inform the design of parallel object database servers. Therefore, in this section we give an overview of the design of parallel relational database servers. This will then allow us to highlight commonalities and differences in the requirements (Section 3) and design options (Section 4) between the two types of parallel database servers. The rest of this section is structured as follows. We begin by describing the architectures of parallel platforms (hardware and operating system) designed to support parallel database servers. Next, we describe methods for exploiting parallelism found in relational database workloads. Throughout this section we will draw on examples from our experience in the design and use of the ICL Goldrush MegaServer [1].

2.1 Parallel Platforms

In this section, we describe the design of parallel platforms (which we define as comprising the hardware and operating system) to meet the requirements of database servers.

We are interested in systems utilising the highly scaleable distributed memory parallel hardware architecture [5] in which a set of computing nodes are connected by a high performance network (Fig. 1). Each node has the architecture of a uniprocessor or shared store multiprocessor - one or more CPUs share the local main memory over a bus. Typically, each node also has a set of locally connected disks for the persistent

storage of data. Connecting disks to each node allows the system to provide both high IO performance, by supporting parallel disk access, and high storage capacity. In most current systems the disks are arranged in a share nothing configuration - each disk is physically connected to only one node. As will be seen, this has major implications for the design of the parallel database server software. It is likely that in future the availability of high bandwidth peripheral interconnects will lead to the design of platforms in which each disk is physically connected to more than one node [11], however this 'shared disk' configuration is not discussed further in this paper as we focus on currently prevalent hardware platform technology. Database servers require large main memory caches for efficient performance (so as to reduce the number of disk accesses) and so the main memories tend to be large (currently 0.25-4GB is typical). Some of the nodes in a parallel platform will have external network connections to which clients are connected. External database clients send database queries to the server through these connections and later receive the results via them. The number of these nodes (which we term Communications Nodes) varies depending on the required performance and availability. In terms of performance, it is important that there are enough Communications Nodes to perform client communication without it becoming a bottleneck, while for availability it is important to have more than one route from a client to a parallel server so that if one fails, another is available.

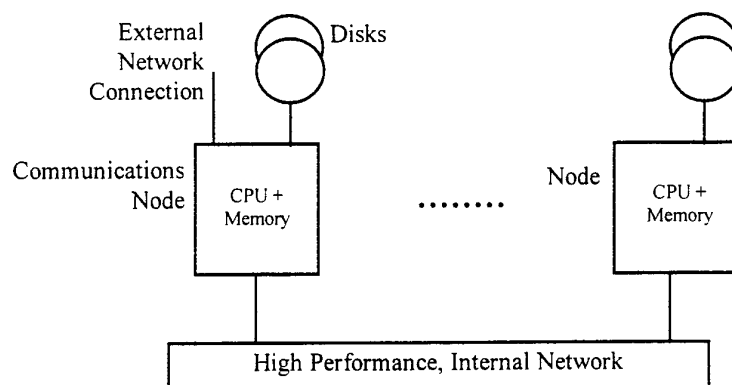


Fig. 1. A Distributed Memory Parallel Architecture

The distributed memory parallel hardware architecture is highly scaleable as adding a node to a system increases all the key performance parameters including: processing power, disk throughput and capacity, and main memory bandwidth and capacity. For a database server, this translates into increased query processing power, greater database capacity, higher disk throughput and a larger database cache.

The design of the hardware must also contribute to the creation of a high availability system. Methods of achieving this include providing component redundancy and the ability to replace failed components through hot-pull & push techniques without having to take the system down [5].

The requirement to support a database server also influences the design of the operating system in a number of ways. Firstly, commercial database server software depends on the availability of a relatively complete set of standard operating system facilities, including support for: file accessing, processes (with the associated inter-process communications) and external communications to clients through standard protocols. Secondly, it is important that the cost of inter-node communications is minimised as this directly affects the performance of a set of key functions including remote data access and query execution. Finally, the operating system must be designed to contribute to the construction of a high availability system. This includes ensuring that the failure of one node does not cause other nodes to fail, or unduly affect their performance (for example through having key services such as distributed file systems delayed for a significant time waiting, until a long time-out occurs, for a response from the failed node).

Experience in a number of projects suggests that micro-kernel based operating systems provide better support than do monolithic kernels for adding and modifying the functionality of the kernel to meet the requirements of parallel database servers [1]. This is for two reasons: they provide a structured way in which new services can be added, and their modularity simplifies the task of modifying existing services, for example to support high-availability [4].

2.2 Exploiting Parallelism

There are two main schemes utilised by parallel relational database servers for query processing on distributed memory parallel platforms. These are usually termed Task Shipping and Data Shipping. They are described and compared in this subsection so that in Section 4 we can discuss their appropriateness to the design of parallel object database servers.

In both Task and Data shipping schemes, the database tables are horizontally partitioned across a set of disks (i.e. each disk stores a set of rows). The set of disks is selected to be distributed over a set of nodes (e.g. it may consist of one disk per node of the server). The effect is that the accesses to a table are spread over a set of disks and nodes, so giving greater aggregate throughput than if the table was stored on a single disk. This also ensures that the table size is not limited to the capacity of a single disk.

In both schemes, each node runs a database server which can compile and execute a query or, as will be described, execute part of a parallelised query. When a client sends an SQL query to one of the Communications Nodes of the parallel server it is directed to a node where it is compiled and executed either serially, on a single node, or in parallel, across a set of nodes. The difference between the two schemes is in the method of accessing database tables as is now described.

2.2.1 Data Shipping

In the data shipping scheme the parallel server runs a distributed filesystem which allows each node to access data from any disk in the system. Therefore, a query running on a node can access any row of any table, irrespective of the physical location of the disk on which the row is stored.

When inter-query parallelism is exploited, a set of external clients generate queries which are sent to the parallel server for execution. On arrival, the Communications Node forwards them to another node for compilation and execution. Therefore the Communication Nodes need to use a load balancing algorithm to select a node for the compilation and execution of each query. Usually, queries in OLTP workloads are too small to justify parallelisation and so each query is executed only on one node. When a query is executed, the database server accesses the required data in the form of table rows. Each node holds a database cache in main memory and so if a row is already in the local cache it can be accessed immediately. However, if it is not in the cache then the distributed filesystem is used to fetch it from disk. The unit of transfer between the disk and cache is a page of rows. When query execution is complete the result is returned to the client.

When a single complex query is to be executed in parallel (intra-query parallelism) then parallelism is exploited within the standard operators used to execute queries: scan, join and sort [12]. For example, if a table has to be scanned to select rows which meet a particular criteria then this can be done in parallel by having each of a set of nodes scan a part of the table. The results from each node are appended to produce the final result. Another type of parallelism, pipeline parallelism, can be exploited in queries which require multiple levels of operators by streaming the results from one operator to the inputs of others.

Because, in the Data Shipping model, any node can access any row of any table the compiler is free to decide for each query both how much parallelism it is sensible to exploit, and the nodes on which it should be executed. Criteria for making these decisions include the granularity of parallelism and the current loading of the server, but it is important to note that these decisions are not constrained by the location of the data on which the query operates. For example, even if a table is only partitioned over the disks of three nodes, a compiler can still choose to execute a scan of that table over eight nodes.

The fact that any node can access data from disks located anywhere in the parallel system has a number of major implications for the design of the parallel database server:

- the distributed filesystem must meet a number of requirements which are not found in typical, conventional distributed filesystems such as NFS. These include high performance access to both local and remote disks, continuous access to data even in the presence of disk and node failures, and the ability to perform synchronous writes to remote disks. Consequently, considerable effort has been expended in this area by developers of parallel database platforms.
- a global lock manager is required as a resource shared by the entire parallel system. A table row can be accessed by a query running on any node of the system.

Therefore the lock which protects that row must be managed by a single component in the system so that queries running on different nodes of the system see the same lock state. Consequently, any node which wishes to take a lock must communicate with the single component that manages that lock. If all locks in a system were to be managed by a single component - a centralised lock manager - then the node on which that component ran would become a bottleneck, reducing the system's scalability. Therefore, parallel database systems usually implement the lock manager in a distributed manner. Each node runs an instance of the lock manager which manages a subset of the locks [1]. When a query needs to acquire or drop a lock it sends a message to the node whose lock manager instance manages that lock. One method of determining the node which manages a lock is to use a hash function to map a lock identifier to a node. All the lock manager instances need to co-operate and communicate to determine deadlocks, as circles of dependencies can contain a set of locks managed by more than one lock manager instance [13].

- distributed cache management is required. A row can be accessed by more than one node and this raises the issue that a row could be cached in more than one node at the same time. It is therefore necessary to have a scheme for maintaining cache coherency. One solution is the use of cache locks managed by the lock manager.

The efficiency of database query execution in a Data Shipping system is highly dependent on the cache hit rate. A cache miss requires a disk access and the execution of filesystem code on two nodes (assuming the disk is remote), which increases response time and reduces throughput. Two techniques can be used to reduce the resulting performance degradation. The first is to, where possible, direct queries which access the same data to the same node. For example all banking queries from one branch could be directed to the same node. This increases cache hit rates and reduces paging: the process by which cached pages have to be moved around the caches of different nodes of the system because the data contained in them is updated by queries running on more than one node. The second technique extends the first technique so that not only are queries operating on the same data sent to the same node, but that node is selected as the one which holds the data on its local disks. The result is that if there is a cache miss then only a local disk access is required. This not only reduces latency, but also increases throughput as less code is executed in the filesystem for a local access than a remote access. This second technique is closely related to Task Shipping, which is now described.

2.2.2 Task Shipping

The key difference between the two Shipping schemes is that in Task Shipping only those parts of tables stored on local disks can be accessed during query evaluation. Consequently, each query must be decomposed into sub-queries, each of which only requires access to the table rows stored on a single node. These sub-queries are then shipped to the appropriate nodes for execution. For a parallel scan operation, this is straightforward to organise, though it does mean that, unlike in the case of Data Shipping, the parallel speed-up is limited by the number of nodes across which the

table is partitioned, unless the data is temporarily re-partitioned by the query. When a join is performed, if the two tables to be joined have been partitioned across the nodes such that rows with matching values of the join attribute are stored on disks of the same node, then the join can straightforwardly be carried out as a parallel set of local joins, one on each node. However, if this is not the case, then at least one of the tables will have to be re-partitioned before the join can be carried out. This is achieved by sending each row of the table(s) to the node in which it will participate in a local join. Parallel sorting algorithms also require this type of redistribution of rows, and so it may be said that the term Task Shipping is misleading because in common situations it is necessary to ship data around the parallel system. However because data is only ever accessed from disk on one node - the node connected to the disk on which it is stored, then some aspects of the system design are simplified when compared to the Data Shipping scheme, viz.:

- there is no requirement for a distributed filesystem because data is always accessed from local disk (however, in order that the system be resilient to node failure, it will still be necessary to duplicate data on a remote node or nodes - see Section 2.3).
- there is no requirement for global cache coherency because data is only cached on one node - that connected to the disk on which it is stored.
- lock management is localised. All the accesses to a particular piece of data are made only from one node - the node connected to the disk on which it is stored. This can therefore run a local lock manager responsible only for the local table rows. However, a 2-phase commitment protocol is still required across the server to allow transactions which have been fragmented across a set of nodes to commit in a co-ordinated manner. Further, global deadlock detection is still required.

An implication of the Task Shipping scheme is that even small queries which contain little or no parallelism still have to be divided into sub-queries if they access a set of table rows which are not all stored on one node. This will be inefficient if the sub-queries have low granularity.

2.3 Availability

If a system is to be highly available then its components must be designed to contribute to this goal. In this section we discuss how the database server software can be designed to continue to operate when key components of a parallel platform fail.

A large parallel database server is likely to have many disks, and so disk failure will be a relatively common occurrence. In order to avoid a break of service for recovery when a disk fails it is necessary to duplicate data on more than one disk. Plexing and RAID schemes can be used for this. In Goldrush, data is partitioned over a set of disk volumes, each of which is duplexed. The two plexes are always chosen to be remote from each other, i.e. held on disks not connected to the same node, so that even if a node fails then at least one plex is still accessible.

If a node fails then the database server running on that node will be lost, but this need not prevent other nodes from continuing to provide a service, all be it with

reduced aggregate performance. Any lock management data held in memory on the failed node becomes inaccessible and so, as this may be required by transactions running on other nodes, the lock manager design must ensure that the information is still available from another node. This can be achieved by plexing this information across the memories of two nodes.

Failure of an external network link may cause the parallel server to loose contact with an external client. In the Goldrush system, the parallel server holds information on alternative routes to clients and monitors the external network connections so that if there is a failure then an alternative route can be automatically chosen.

3 Parallel ODBMS Requirements

In this section we describe and justify the requirements which we believe a parallel ODBMS server should meet. A major source of requirements is derived from our experience with the design and usage of the ICL Goldrush MegaServer [1, 5, 6] which embodies a number of the functions and attributes that will also be required in a parallel ODBMS. However, it is also the case that there are interesting differences between the requirements for a parallel RDBMS, such as Goldrush, and a parallel ODBMS. In particular, we envisage a parallel ODBMS as a key component for building high performance distributed applications as it has attributes not found in alternative options including: performance, availability, rich interfaces for accessing information (including querying), and transactional capability to preserve database integrity. However, if a parallel ODBMS is to fulfill its potential in this area then we believe that there are a set of requirements which it must meet, and these are discussed in this section.

The rest of this section is structured as follows. We first describe the overall system requirements, before examining the non-functional requirements of performance and availability.

3.1 Systems Architecture

Our main requirement is for a server which provides high performance to ODMG compatible database clients by exploiting both inter-query and intra-query parallelism. This requires a system architecture similar to that described for RDBMS in Section 2. Client applications may generate OQL queries which are sent for execution to the parallel server (via the Communications Nodes). To support intra-query parallelism the server must be capable of executing these queries in parallel across a set of nodes. The server must also support inter-query parallelism by simultaneously executing a set of OQL queries sent by a set of clients. Clients may also map database objects into program objects in order to (locally) access their properties and call methods on them. Therefore, the server must also satisfy requests from clients for copies of objects in the database. The roles of the parallel system, serving clients, are summarised by Figure 2.

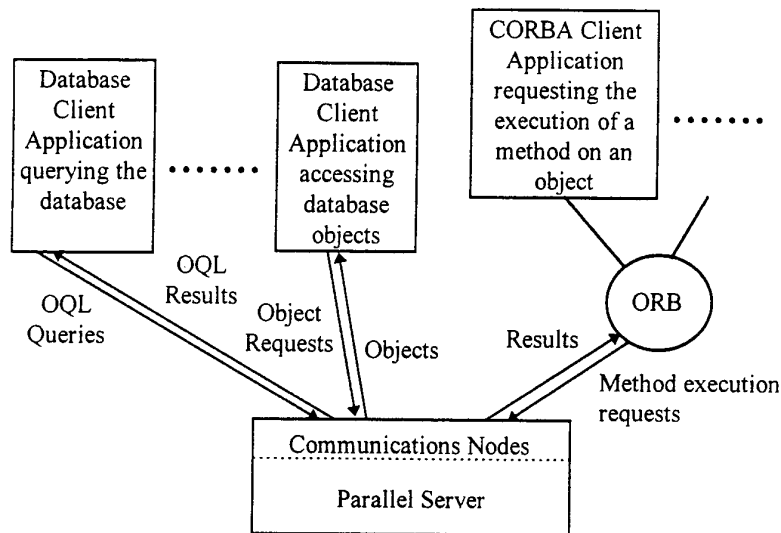


Fig. 2. Clients of the Parallel ODBMS Server

The architecture does not preclude certain clients from choosing to execute OQL queries locally, rather than on the parallel server. This would require the client to have local OQL execution capability, but the client would still need to access the objects required during query execution from the server. Reasons for doing this might be to exploit a particularly high performance client, or to remove load from the server so that it can spend more of its computational resources on other, higher priority queries.

Where the ODBMS is to act as a component in a distributed system, we require that it can be accessed through standard CORBA interfaces [14]. The ODMG standard proposes that a special component - an Object Database Adapter (ODA) - is used to connect the ODBMS to a CORBA Object Request Broker (ORB) [3]. This allows the ODBMS to register a set of objects as being available for external access via the ORB. CORBA clients can then generate requests for the execution of methods on these objects. The parallel ODBMS services these requests, returning the results to the clients. In order to support high throughput, the design of the parallel ODBMS must ensure that the stream of requests through the ODA is executed in parallel.

3.2 Performance

Whilst high absolute performance is a key requirement, experience with the commercial usage of parallel relational database servers has shown that scalability is a key attribute. It must be possible to add nodes to increase the server's performance for both inter-query and intra-query parallelism. In Section 2, we described how the

distributed memory parallel architecture allows for scalability at the hardware platform level, but this is only one aspect of achieving database server scalability. Adding a node to the system provides more processing power which may be used to speed-up the evaluation of queries. However for those workloads whose performance is largely dependent on the performance of object access, then adding a node will have little or no immediate effect. In the parallel ODBMS, sets of objects will be partitioned across a set of disks and nodes (c.f. the design of a parallel RDBMS as described in Section 2), and so increasing the aggregate object access throughput requires re-partitioning the sets of objects across a larger number of disks and nodes.

Experience with serial ODBMS has shown that the ability to cluster on disk objects likely to be accessed together is important for reducing the number of disk accesses and so increasing performance [15]. Indeed, the ODMG language bindings provide versions of object constructors which allow the programmer to specify that a new object should be clustered with an existing object [3]. Over time, the pattern of access to objects may change, or be better understood (due to information provided by performance monitoring), and so support for re-clustering is also required to allow performance tuning.

As will be seen in Section 4, the requirement to support the re-partitioning and re-clustering of objects has major implications for the design of a parallel ODBMS.

3.3 Performance Management

This section describes requirements which relate to the use of a database as a resource shared among a set of services with different performance requirements.

The computational resources (CPU, disk IOs, etc.) made available by high performance systems are expensive - customers pay more per unit resource on a high performance system than they do on a commodity computer system. In many cases, resources will not be completely exhausted by a single service but if the resources are to be shared then mechanisms are needed to control this sharing.

An example of the need for controlled resource sharing is where a database runs a business-critical OLTP workload which does not utilise all the server's resources. It may be desirable to run other services against the same database, for example in order to perform data mining, but it is vital that the performance of the business-critical OLTP service is not affected by the loss of system resources utilised by other services. In a distributed memory parallel server, the two services may be kept apart on non-intersecting sets of nodes in order to avoid conflicts in the sharing of CPU resources. However, they will still have to share the disks on which the database is held. A solution might be to analyse the disk access requirements of the two services and try to ensure that the data is partitioned over a sufficiently large set of nodes and disks so as to be able to meet the combined performance requirements of both services. However this is only possible where the performance requirements of the two services are relatively static and therefore predictable. This may be true of the OLTP service, but the performance of a complex query or data mining service may not be predictable. A better solution would be to ensure that the OLTP service was allocated the share of the

resources that it required to meet its performance requirements, and only allow the other service the remainder.

A further example of the importance of controlled resource sharing is where parallel ODBMS act as repositories for persistent information on the Internet and intranets. If information is made available in this way then it is important that the limited resources of the ODBMS are shared in a controlled way among clients. For example there is a danger of denial of service attacks in which all the resources of system are used by a malicious client. Similarly, it would be easy for a non-malicious user to generate a query which required large computational resources, and so reduced the performance available to other clients below an acceptable level. Finally, it may be desirable for the database provider to offer different levels of performance to different classes of clients depending on their importance or payment.

The need for controlled sharing of resources has long been recognised in the mainstream computing world. For example, mainframe computer systems have for several decades provided mechanisms to control the sharing of their computational resources among a set of users or services. Some systems allow CPU time and disk IOs to be divided among a set of services in any desired ratio. Priority mechanisms for allocating resources to users and services may also be offered. Such mechanisms are now also provided for some shared memory parallel systems.

Unfortunately, the finest granularity at which these systems control the sharing of resources tends to be at an Operating System process level. This is too crude for database servers, which usually have at their heart a single, multi-threaded process which executes the queries generated by a set of clients (so reducing switching costs when compared to an implementation in which each client has its own process running on the server). Therefore, in conclusion, we believe that the database server must provide its own mechanisms to share resources among a set of services in a controlled manner if expensive parallel system resources are to be fully utilised, and if object database servers are to fulfil their potential as key components in distributed systems.

3.4 Availability

The requirements here are identical to those of a parallel RDBMS. The parallel ODBMS must be able to continue to operate in the presence of node, disk and network failures. It is also important that availability is considered when designing database management operations which may effect the availability of the database service. These include: archiving, upgrading the server software and hardware, and re-partitioning and re-clustering data to increase performance. Ideally these should all be achievable on-line, without the need to shut-down the database service, but if this is not possible then the time for which the database service is down should be minimised.

A common problem causing loss of availability during management operations is human error. Manually managing a system with up to tens of nodes, and hundreds of disks is very error prone. Therefore tools to automate management tasks are important for reducing errors and increasing availability.

3.5 Hardware Platform

The *Polar* project has access to a parallel database platform, an ICL Goldrush MegaServer [1], however we are also investigating alternatives. Current commercial parallel database platforms use custom designed components (including internal networks, processor-network interfaces and cabinetry), which leads to high design, development and manufacturing costs, which are then passed on to users. Such systems generally have a cost-performance ratio that is significantly higher than that of commodity, uniprocessor systems. We are investigating a solution to this problem and have built a parallel machine, the Affordable Parallel Platform (APP), entirely from standard, low-cost, commodity components - high-performance PCs interconnected by a high throughput, scaleable ATM network. Our current system consists of 13 nodes interconnected by 155Mbps ATM. The network is scaleable as each PC has a full 155Mbps connection to the other PCs. This architecture has the interesting property that high-performance clients can be connected directly to the ATM switch giving them a very high bandwidth connection to the parallel platform. This may, for example, be advantageous if large multimedia objects must be shipped to clients.

Our overall aim in this area is to determine whether commodity systems such as the APP can compete with custom parallel systems in terms of database performance. Therefore we require that the database server design and implementation is portable across, and tuneable for, both types of parallel platform.

4 Parallel ODBMS Design

In this section we consider issues in the design of a parallel ODBMS to meet the requirements discussed in the last section. We cover object access in the most detail as it is a key area in which existing parallel RDBMS and serial ODBMS designs do not offer a solution. Cache coherency, performance management and query processing are also discussed.

4.1 Object Access

Objects in an object database can be individually accessed through their unique identifier. This is a property not found in relational databases and so we first discuss the issues in achieving this on the parallel server (we leave discussion of accessing collections of objects until Section 4.3).

As described earlier, the sets of objects stored in the parallel server are partitioned across a set of disks and nodes in order to provide a high aggregate access throughput to the set of objects. When an object is created, a decision must be made as to the node on which it will be stored. When a client needs to access a persistent object then it must send the request to the node holding the object.

In the rest of this section we discuss in more detail the various issues in the storage and accessing of objects in a parallel ODBMS as these differ significantly from the design of a serial ODBMS or a parallel RDBMS. Firstly we discuss how the Task Shipping and Data Shipping mechanisms described in Section 2 apply to a parallel ODBMS. We then discuss possible options for locating and accessing persistent objects.

4.1.1 Task Shipping vs. Data Shipping

In a serial ODBMS, the database server stores the objects and provides a run-time system which allows applications running on external clients to transparently access objects. Usually a page server interface is offered to clients, i.e. a client requests an object from the server which responds with a page of objects [7]. The page is cached in the client because it is likely that the requested object and others in the same page will be accessed in the near future. This is a Data Shipping architecture, and has the implication that the same object may be cached in more than one client, so necessitating a cache coherency mechanism. If Data Shipping was also adopted within the parallel server then nodes executing OQL would also access and cache objects from remote nodes.

We now consider if the alternative used in some parallel RDBMS - a Task Shipping architecture - is a viable option for a parallel ODBMS. There are two computations which can be carried out on objects: executing a method on an object and accessing a property of an object. These computations can occur either in an application running on an external client, or during OQL execution on a node of the parallel server.

In order to implement task shipping, the external clients, and nodes executing OQL would not cache remote objects and process them locally. Instead, they would have to send a message to the node which stored the object requesting that it perform the computation (method execution or property access) locally and return the result. In this way, the client does not cache objects, and so cache coherency is not an issue. However Task Shipping in object database servers does have two major drawbacks. Firstly, the load on the server is increased when compared to the data shipping scheme. This means that a server is likely to be able to support fewer clients. Secondly, the latency of task shipping - the time between sending the task to the server and receiving the response - is likely to result in performance degradation at the client when compared to the data shipping scheme in which the client can operate directly on the object without any additional latency once it has been cached. Many database clients are likely to be single application systems, and so there will not be other processes to run while a computation on an object is being carried out on the parallel server - consequently the CPU will be idle during this time. Further, the user of an interactive client application will be affected by the increase in response time caused by the latency.

The situation is slightly different for the case of OQL query execution on the nodes of the server under a Task Shipping scheme. When query execution is suspended while a computation on an object is executed remotely, it is very likely that there will be other work to do on the node (for example, servicing requests for operations on objects and executing other queries) and so the CPU will not be idle. However, it is

likely that the granularity of parallel processing (the ratio of computation to communication) will be low due to the frequent need to generate requests for computations on remote objects. Therefore, although the node may not be idle, it will be executing frequent task switches and sending/receiving requests for computations on objects. The effect of this is likely to be a reduction in throughput when compared to a data shipping scheme with high cache hit rates which will allow a greater granularity of parallelism.

For these reasons, we do not further consider a task shipping scheme, and so must address the cache coherence issue (Section 4.2). Figure 3 shows the resulting parallel database server architecture. The objects in the database are partitioned across disks connected to the nodes of the system, taking into account clustering. Requests to access objects will come from both: applications running on external database clients; and the nodes of the parallel server which are executing queries or processing requests for method execution from CORBA clients. The database client applications are supported by a run-time system (RTS) which maintains an object cache. If the application accesses an object which is not cached then an object request is generated and sent to a Communication Node (CN) of the parallel server. The CN uses a Global Object Access module to determine the node which has the object stored on one of its local disks. The Remote Object Access component forwards the request to this node where it is serviced by the Local Object Access component which returns the page containing the object back to the client application RTS via the CN.

If the client application issues an OQL query then the client's database RTS sends it to a CN of the parallel server. From there it is forwarded to a node for compilation. This generates an execution plan which is sent to the Query Execution components on one or more nodes for execution. The Query Execution components on each node are supported by a local run time system, identical to that found on the external database clients, which accesses and caches objects as they are required, making use of the Global Object Access component to locate the objects. The question of how objects are located is key, and is discussed in the next section.

If there are CORBA based clients generating requests for the execution of object methods then the requests will be directed (by the ORB) to a CN. In order to maximise the proportion of local object accesses (which are more efficient than remote object accesses) then the CN will route the request to the node which stores the object for execution. However, there will be circumstances in which this will lead to an imbalance in the loading on the nodes, for example if one object frequently occurs in the requests. In these circumstances, the CN may choose to direct the request to another, more lightly loaded, node.

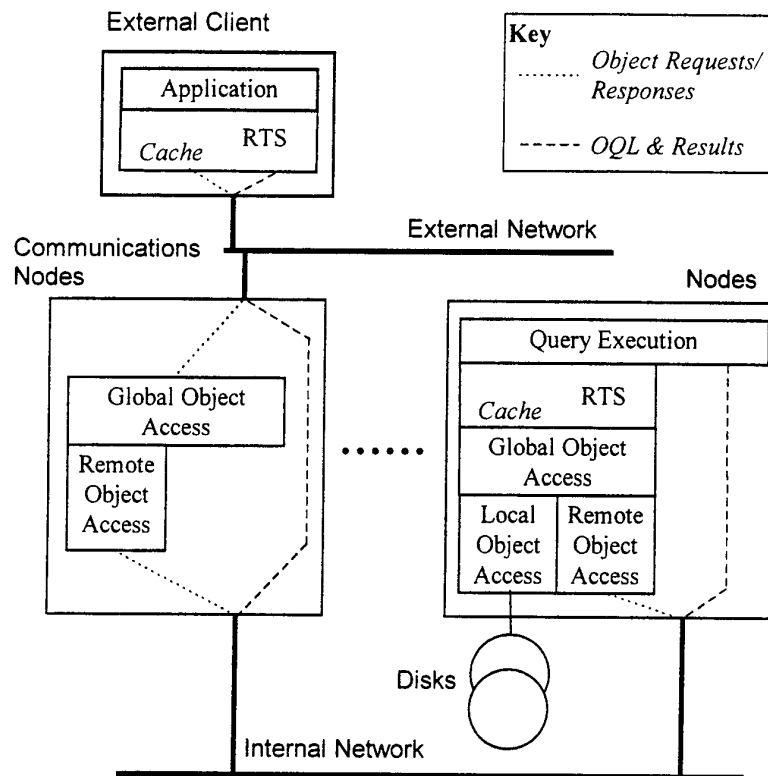


Fig. 3. Database Server System Architecture

4.1.2 Persistent Object Access

In this section we describe the major issues in locating objects. These include the structure of object identifiers (OIDs) and the mechanism by which an OID is mapped to a persistent object identifier (PID). A PID is the physical address of the object, i.e. it includes information on the node, disk volume, page and offset at which the object can be accessed.

Based on the requirements given in Section 3, the criteria which an efficient object access scheme for a parallel server must meet are:

- a low OID to PID mapping cost. This could be a significant contribution to the total cost of an object access and is particularly important as the Communications Nodes (which are only a subset of the nodes in the system) must perform this mapping for all the object requests from external ODBMS and CORBA clients. We want to minimise the number of CNs as they will be more expensive than other nodes, and because the maximum number of CNs in a parallel platform may be limited for

physical design reasons. The OID to PID mapping will also have to be performed on the other nodes when the Query Execution components access objects.

- minimising the number of nodes involved in an object access as each inter-node message contributes to the cost of the access.
- ability to re-cluster objects if access patterns change.
- ability to re-partition sets of objects over a different number of nodes and disks to meet changing performance requirements.
- a low object allocation cost.

We now consider a set of possible options for object access and consider how they match up against these criteria.

4.1.2.1 Physical OID schemes

In this option, the PID is directly encoded in the OID. For example, the OID could have the structure:

Node	Disk Volume	Page	Offset
------	-------------	------	--------

The major advantage of this scheme is that the cost of mapping the OID to the PID is zero. The Communications Nodes (CNs) only have to examine the OIDs within incoming object requests from external clients and forward the request to the node contained in the OID. This is just a low-level routing function and could probably be most efficiently performed at a low level in the communications stack. Therefore, the load on a CN incurred by each external object access will be low. Similarly, object accesses generated on a node by the Query Execution component can be directed straight to the node which owns the object. Therefore this scheme does minimise the number of nodes involved in an access. However, because the OID contains the exact physical location of the object, it is not possible to move an object to a different page, disk volume or node. This rules out re-clustering or re-partitioning.

When an object is to be created then the first step is to select a node. This could be done by the CN using a round-robin selection scheme to evenly spread out the set of objects. The object is then created on the chosen node. Therefore object creation is a low cost operation.

Therefore, this scheme will perform well in a stable system but as it does not allow re-partitioning or re-clustering, it is likely that over time the system will become detuned as the server's workload changes.

We now consider three possible variations of the scheme which attempt to overcome this problem:

Indirections. The OID scheme used by O2 [7] includes a physical volume number in the OID, and so runs into similar problems if an Object is to be moved to a different volume. The O2 solution is to place an indirection at the original location of the object pointing to the new location of the object. Therefore, potentially two disk accesses might be incurred and (in a parallel server) two nodes might be involved in an object access. Also, this is not a viable solution for re-partitioning a set of objects across a larger number of disks and nodes in order to increase the aggregate throughput to those objects. The initial accesses to all the objects will still have to go to the

indirections on the original disks/nodes and so they will still be a bottleneck. For this reason we disregard this variant as a solution.

Only include the Physical Node in the OID. In this variant, the OID structure would contain the physical node address, and then a logical address within the node, i.e. :

Node	Logical address within node
------	-----------------------------

The logical address would be mapped to the physical address at the node; methods for this include hash tables and B-trees (the Direct Mapping scheme[16] is however not an option as it is not scaleable: once allocated, the handles which act as indirections to the objects cannot be moved). This would allow re-clustering within a node by updating the mapping function, and re-clustering across nodes using indirections, but it would not support re-partitioning across more nodes as the Node address is still fixed by the OID. Therefore we disregard this variant as a solution.

Update OIDs when an Object is Moved. If this could be implemented efficiently then it would have a number of advantages. Objects could be freely moved within and between nodes for re-clustering and re-partitioning, but because their OID would be updated to contain their new physical address then the mapping function would continue to have zero cost, and no indirections would be required. However, such a scheme would place restrictions on the external use of OIDs. Imagine a scenario in which a client application acquires an OID and stores it. The parallel ODBMS then undergoes a re-organisation which changes the location of some objects, including the one whose OID was stored by the client application. That OID is now incorrect and points to the wrong physical object location. A solution to this problem is to limit the temporal scope of OIDs to a client's database session. The ODMG standard allows objects to be named, and provides a mapping function to derive the object's OID from its name. It could be made mandatory for external database clients to only refer to objects by name outside of a database session. Within the session the client would use the mapping function to acquire the OID of an object. During the session, this OID could be used, but when the session ended then the OID would cease to have meaning and at the beginning of subsequent sessions OIDs would have to be re-acquired using the name mapping function. This protocol leaves the database system free to change OIDs while there are no live database sessions running. When an object is moved then the name mapping tables would be updated. Any references to the object from other objects in the database would also have to be updated. If each reference in a database is bi-directional then this process is simplified, as all references to an object could be directly accessed and updated. The ODMG standard does permit unidirectional references but these can still be implemented bi-directionally. Another option is to scan the whole database to locate any references to the object to be moved. In a large database then this could be prohibitively time-consuming. An alternative is the Thor indirection based scheme [10], in which, over a period of time, references are updated to remove indirections.

A name based external access scheme could be used for CORBA client access to objects. Objects could be registered with an ORB by name. Therefore, incoming requests from CORBA based clients would require the object name to be mapped to

an OID at a CN. A CORBA client could also navigate around a database so long as it started from a named object, and provided that the OIDs of any other objects that it reached were registered with the ORB. However, if the client wished to store a reference to an object for later access (outside the current session) then it would have to ask the ODBMS to create a name for the object. The name would be stored in the mapping tables on the CNs, ready for subsequent accesses. This procedure can, for example, be used to create interlinked, distributed databases in which references in one database can refer to objects in another.

4.1.2.2 Logical OID schemes

At the other extreme from the last scheme is one in which the OID is completely logical and so contains no information about the physical location of the object on disk. Consequently, each object access requires a full logical to physical mapping to determine the PID of the object. Methods of performing this type of mapping have been extensively studied with respect to a serial ODBMS [16], however their appropriateness for a parallel system requires further investigation.

As stated above, the logical to physical mapping itself can be achieved by a number of methods including hash tables and B-trees. Whatever method is used for the mapping, the Communication Nodes will have to perform the mapping for requests received by external clients. This may require disk accesses as the mapping table for a large database will not fit into main memory. Therefore there will be a significant load on the CNs, and it will be necessary to ensure that there are enough CNs configured in a system so that they are not bottlenecks. Similarly, the nodes executing OQL will also have to perform the expensive OID to PID mapping. However, once the mapping has been carried out then the request can be routed directly to the node on which the object is located.

As all nodes will need to perform the OID to PID mapping, then each will need a complete copy of the mapping information. We rule out the alternative solution of having only a subset of the nodes able to perform this mapping due to the extra message passing latency that this would add to each object access. This has two implications: each node needs to allocate storage space to hold the mapping; and whenever a new object is created then the mapping information in all nodes needs to be updated, which results in a greater response time, and worse throughput for object creation than was the case for the Physical OID scheme described in the last section.

With this scheme, unlike the Physical OID scheme, names are not mandatory for storing external references to objects. As OIDs are unchanging, they can be used both internally as well as externally. This removes the need for CNs to map names to OIDs for incoming requests from CORBA clients.

4.1.2.3 Virtual Address Schemes

There have been proposals to exploit the large virtual address spaces available in some modern processors, in object database servers, both parallel [17] and serial [18]. Persistent objects are allocated into the virtual address space and their virtual address is used as their OID. This has the benefit of simplifying the task of locating a cached

object in main memory. However an OID to PID mapping mechanism is still required when a page fault occurs and the page containing an object must be retrieved from disk. The structure of a typical virtual address is:

Segment	Page	Offset
---------	------	--------

In a parallel ODBMS, when a page fault occurs then the segment and page part of the address is mapped to a physical page address. The page is fetched, installed in real memory and the CPU's memory management unit tables are configured so that any accesses to OIDs in that page are directed to the correct real memory address.

The key characteristic of this scheme is that the OID to PID mapping is done at the page level, i.e. in the [Segment, Page, Offset] structure shown above only the Segment and Page fields are used in determining the physical location of the object. Therefore it is possible to re-partition sets of objects by changing the mapping, but it is not possible to re-cluster objects within pages unless changing OIDs, or indirections are supported (as described in Section 4.1.2.1).

The OID to PID mapping can be done using the logical or physical schemes described earlier, with all their associated advantages and disadvantages. However, it is worth noting that if a logical mapping is used then the amount of information stored will be smaller than that required for the logical OID scheme of Section 4.1.2.2 as only pages, rather than individual objects must be mapped.

4.1.2.4 Logical Volume Scheme

None of the above schemes is ideal, and so we have also investigated the design of an alternative scheme which allows the re-partitioning of data without the need to update addresses nor incur the cost of an expensive logical to physical mapping in order to determine the node on which an object is located. In describing the scheme we will use the term *volume set* to denote the set of disk volumes across which the objects in a class are partitioned. OIDs have the following structure:

Class	Logical Volume	Body
-------	----------------	------

The *Class* field uniquely identifies the class of the object. As will be described in detail later, the *Logical Volume* field is used to ensure that objects which must be clustered are always located in the same disk volume, even after re-partitioning. The size of this field is chosen so that its maximum value is much greater than the maximum number of physical volumes that might exist in a parallel system.

Each node of the server has access to a definition of the Volume Set for each class. This allows each node, when presented with an OID, to use the *Class* field of the OID to determine the number and location of the physical volumes in the Volume Set of the class to which the object belongs.

If we denote the cardinality of Volume Set *V* as $|V|$, then the physical volume on which the object is stored is calculated cheaply as:

$$\text{Physical Volume} = V[\text{LogicalVolume modulus } |V|]$$

where the notation $V[i]$ denotes the i 'th Volume in the Volume Set V . A node wishing to access the object uses this information to forward the request to the node on which the object is held. On that node, the unique serial number for the object within the physical volume can be calculated as:

$$\text{Serial} = (\text{Body} \text{ div } |V|) + |\text{Body}| * (\text{LogicalVolume} \text{ div } |V|)$$

where: $a \text{ div } b$ is the whole number part of the division of a by b , and $|\text{Body}|$ is the number of unique values that the body field can take.

The Serial number can then be mapped to a PID using a local logical to physical mapping scheme.

If it is necessary to re-partition a class then this can be done by creating a new volume set with a different cardinality and copying each object from the old volume set into the correct volume of the new volume set using the above formula. Therefore a class of objects can be re-partitioned over an arbitrary set of volumes without having to change the OIDs of those objects.

The main benefit of the scheme is that objects in the same class with the same *Logical Volume* field will always be mapped to the same physical volume. This is still true even after re-partitioning, and so ensures that objects clustered before partitioning can still be clustered afterwards. Therefore, when objects which must be clustered together are created they should be given OIDs with the same value in the *Logical Volume* field. To ensure that the OID is unique they must have different values for their *Body* fields.

Where it is desirable to cluster objects of different classes then this can be achieved by ensuring that both classes share the same volume set, and that objects which are to be clustered share the same *Logical Volume* value. This ensures that they are always stored on the same physical volume.

When an external client application needs to create a new object then it sends the request, including the class of the object, to a Communication Node. These hold information on all the volume sets in the server, and so can select one physical volume from that set (using a round-robin or random algorithm). The CN then forwards the create object request to the node that contains the chosen physical volume, and the OID is then generated on that node as follows. Each node contains the definition of the volume set for each class. The required values of the Class and Logical Volume fields of the object will have been sent to the node by the CN. The node also keeps information on the free *Body* values for each of the logical volumes which map to a physical volume connected to that node. The node can therefore choose a free *Body* value for the Logical Volume chosen by the CN. This completes the OID. The object is then allocated to a physical address and the local OID mapping information updated to reflect this new OID to PID mapping.

The creation algorithm is different in cases where it is desirable to create a new object which is to be clustered with an existing object. The new object must be created with the same *Logical Volume* field setting as the existing object so that it will always be stored on the same Physical Volume as the existing object, even after a re-partition. Therefore it is this Logical Volume value which is used to determine the node on which the object is created. On that node the Body field of the new object will be

selected, and the object will be allocated into physical disk storage clustered with the existing object (assuming that space available).

This scheme therefore allows us to define a scaleable, object manager in which it is possible to re-partition a set of objects in order to meet increased performance requirements. Existing clustering can be preserved over re-partitioning of the data, however if re-clustering is required then it would have to be done by either using indirections or by updating OIDs. Determining the node on which an object resides is cheap and this should reduce the time required on the CNs to process incoming object access requests. The vast bulk of the information required to map an OID to a PID is local to the node on which the object is stored, so reducing the total amount of storage in the system required for OID mapping, and reducing the response time and execution cost of object creation.

4.1.2.5 Summary

The above schemes all have advantages and disadvantages:

- The Physical OID scheme has a low run time cost but our requirements for re-clustering and re-partitioning can only be met if OIDs can be changed. However, the use of names as permanent identifiers does permit this.
- The Logical OID system supports both re-partitioning and re-clustering. However it is expensive both for object creation and in terms of the load it places on the CNs.
- The Virtual Address scheme supports re-partitioning, but re-clustering is only possible through indirections or by changing OIDs. Creating an object is costly when a new page has to be allocated as it requires an updating of the OID to PID mapping on all nodes. Also, the cost of mapping OIDs to PIDs for accesses from external clients may place a significant load on the CNs.
- The Logical Volume scheme may be a reasonable compromise between completely logical and completely physical schemes. However indirections or changeable OIDs are required for re-clustering.

We are therefore investigating these schemes further in order to perform a quantitative comparison between them.

4.2 Concurrency Control and Cache Management

Concurrency control is important in any system in which multiple clients can be accessing an object simultaneously. In our system, each node runs an Object Manager which is responsible for the set of local objects including: concurrency control when those objects are accessed; and the logging mechanisms required to ensure that it is possible to recover of the state of those objects after a node or system failure.

In the system we have described there are three types of clients of the Object Managers: database applications running on external clients; Query Execution components on the parallel server nodes; and, also on the nodes, object method calls instigated by CORBA based clients. Therefore there are caches both on the nodes of the server, and in the external database clients. More than one client may require access to a particular object and so concurrency control and cache coherency mechanisms must manage the fact that an object may be held in more than one cache.

This situation is not unique to parallel systems, and occurs on any client-server object database system which supports multiple clients. A number of concurrency control/cache coherency mechanisms have been proposed for this type of system [19], and there appears to be nothing special about parallel systems which prevents one of these from being adopted.

4.3 Query Execution

In a parallel system we wish to be able to improve the response time of individual OQL queries by exploiting intra-query parallelism. In this section we discuss some of the issues in achieving this.

An OQL query generated by a client will be received first by a Communications Node which will use a load balancing scheme to choose a node on which to compile the query. Compilation generates an execution plan: a graph whose nodes are operations on objects, and whose arcs represent the flow of objects from the output of one operation to the input of another. As described earlier, the system we propose supports data shipping but for the reasons discussed in Section 2.2.1, it will give performance benefits if operations are carried out on local rather than remote objects where possible. To achieve this it is necessary to structure collections which are used to access objects (e.g. extents) so as to allow computations to be parallelised on the basis of location. One option [17] is to represent collections of objects through two-level structures. At the top level is a collection of sets of objects with one set per node of the parallel system. Each set contains only the objects held on one node. This allows computations on collections of objects to be parallelised in a straightforward manner: each node will run an operation which processes those objects stored locally. However, in our data shipping system if the objects in a collection are not evenly partitioned over the nodes then it is still possible to partition the work of processing the objects among the nodes in a way which is not based on the locations of the objects. One advantage of the Logical Volume OID mapping scheme described in Section 4.1.2.4 is that it removes the need to explicitly maintain these structures for class extents as the existing OID to PID mapping information makes it possible to locate all the objects in a given class stored on the local node.

A major difference between object and relational databases is that object database queries can include method calls. This has two major implications.

Firstly, as the methods must be executed on the nodes of the parallel server a mechanism is required to allow methods to be compiled into executable code which runs on the server. This is complicated by the fact that the ODMG standard allows methods to be written in a number of languages, and so either the server will have to provide environments to compile and execute each of these languages, or alternatively client applications using the system will have to be forced to specify methods only in those languages which the server can support. Further, the architecture that we have proposed in this paper has client applications executing method calls in navigational code on the client. Therefore executable versions of the code must be available both in the server (for OQL) and on the clients. This introduces risks that the two versions of

the code will get out of step. This may, for example, occur if a developer modifies and recompiles the code on a client but forgets to do the same on the server. Also a malicious user might develop, compile and execute methods on the client specifically so as to access object properties not available through the methods installed in the server. Standardising on Java or another interpreted language for writing methods might appear to be a solution to these problems because they have a standard, portable executable representation to which a method could be compiled once and stored in the database before execution on either a client or server. However, the relatively poor performance of interpreted languages when compared to a compiled language such as C++ is a deterrent because the main reason for utilising a parallel server is to provide high performance.

Secondly, the cost of executing a method may be a significant component of the overall performance of a database workload, and may greatly vary in cost depending on its arguments. This could make it difficult to statically balance the work of executing a query over a set of nodes. For example, consider an extent of objects that is perfectly partitioned over a set of nodes such that each node contains the same number of objects. A query is executed which applies a method to each object but not all method calls take the same time, and so, if the computation is partitioned on the basis of object locality, some nodes may have completed their part of the processing and be idle while others are still busy. Therefore it may be necessary to adopt dynamic load balancing schemes which delay decisions on where to execute work for as long as possible (at run-time) so as to try to evenly spread the work over the available nodes. For workloads in which method execution time dominates and not all nodes are fully utilised (perhaps because the method calls are on a set of objects whose cardinality is less than the number of nodes in the parallel system) then it may be necessary to support intra-method parallelism, in which parallelism is exploited within the execution of a single method. This would require methods to be written in a language which was amenable to parallelisation, rather than a serial language such as C++ or Java. One promising candidate is UFO [20], an implicit parallel language with support for objects.

4.4 Performance Management

Section 3 outlined the requirements for performance management in a parallel database server. In this section we describe how we intend to meet these requirements.

In recent years, there has been an interest in the performance management of multimedia systems [21]. When a client wishes to establish a session with a multimedia server, the client specifies the required Quality of Service and the server decides if it can meet this requirement, given its own performance characteristics and current workload. If it can, then it schedules its own CPU and disk resources to ensure that the requirements are met. Ideally, we would like this type of solution for a parallel database server, but unfortunately it is significantly more difficult due to the greater complexity of database workloads. The Quality of Service requirements of the client of a multimedia server can be expressed in simple terms (e.g. throughput and jitter).

and the mapping of these requirements onto the usages of the components of the server is tractable. In contrast, whilst the requirements of a database workload may be specified simply, e.g. response time and throughput, in many cases the mapping of the workload onto the usage of the resources of the system cannot be easily predicted. For example, the CPU and disk utilisation of a complex query are likely to depend on the state of the data on which the query operates.

We have therefore decided to investigate a more pragmatic approach to meeting the requirements for performance management in a parallel database server which is based on priorities [11]. When an application running on an external client connects to the database then that database session will be assigned a priority by the server. For example, considering the examples given in Section 3, the sessions of clients in an OLTP workload will have high priority, the sessions of clients generating complex queries will have medium priority while the sessions of agents will have low priority. Each unit of work - an object access or an OQL query - sent to the server from a client will be tagged with the priority and this will be used by the disk and CPU schedulers. If there are multiple units of work, either object accesses or fragments of OQL available for execution, then they will be serviced in the order of their priority, while pre-emptive round-robin scheduling will be used for work of the same priority.

5 Conclusions

The aim of the *Polar* project is to investigate the design of a parallel, ODMG compatible ODBMS. In this paper we have highlighted the system requirements and key design issues, using as a starting point our previous experience in the design and usage of parallel RDBMS. We have shown that differences between the two types of database paradigms lead to a number of significant differences in the design of parallel servers. The main differences are:

- rows in RDBMS tables are never accessed individually by applications, whereas objects in an ODBMS can be accessed individually by their unique OIDs. The choice of OID to PID mapping is therefore very important. A number of schemes for structuring OIDs and mapping them to PIDs were presented and compared. None is ideal, and further work is needed to quantify the differences between them.
- there are two ways to access data held in an object database: through a query language (OQL), and by directly mapping database objects into client application program objects. In a relational database, the query language (SQL) is the only way to access data. A major consequence of this difference is that parallel RDBMS can utilise either task shipping or data shipping, but task shipping is not a viable option for a parallel ODBMS with external clients.
- object database queries written in OQL can contain arbitrary methods (unlike RDBMS queries) written in one of several high level programming languages. Mechanisms are therefore required in a parallel ODBMS to make the code of a method executable on the nodes of the parallel server. This will complicate the process of replacing an existing serial ODBMS, in which methods are only

executed on clients, with a parallel server. It also raises potential issues of security and the need to co-ordinate the introduction of method software releases. Further, as methods can contain arbitrary code, estimating execution costs is difficult and dynamic load-balancing, to spread work evenly over the set of nodes, may be required.

We have also highlighted the potential role for parallel ODBMS in distributed systems and described the corresponding features that we believe will be required. These include performance management to share the system resources in an appropriate manner, and the ability to accept and load-balance requests from CORBA clients for method execution on objects.

5.1 Future Work

Having carried out the initial investigations described in this paper we are now in a position to explore the design options in more detail through simulation and the building of a prototype system. We are building the prototype in a portable manner so that, as described in Section 3 we can compare the relative merits of a custom parallel machine and one constructed entirely from commodity hardware.

Our investigations so far have also raised a number of areas for further exploration, and these may influence our design in the longer term:

- More sophisticated methods of controlling resource usage in the parallel server are needed so as to make it possible to guarantee to meet the performance requirements of a workload. The solution based on priorities, described in Section 3, does not give as much control over the scheduling of resources as is required to be able to guarantee that the individual performance requirements of a set of workloads will all be met. Therefore we are pursuing other options based on building models of the behaviour of the parallel server and the workloads which run on it. Each workload is given the proportion of the systems resources that it needs to meet its performance targets.
- If it is possible to control the resource usage of database workloads then, in the longer term, it would be desirable to extend the system to support continuous media. Currently multimedia servers (which support continuous media) and object database servers have different architectures and functionality, but there would be many advantages in unifying them, for example to support the high-performance, content-based searching of multimedia libraries.
- The ODMG standard does not define methods of protecting access to objects equivalent to that found in RDBMS. Without such a scheme, there is a danger that clients will be able to access and update information which should not be available to them. This will be especially a problem if the database is widely accessible, for example via the Internet. Work is needed in this area to make information more secure from both malicious attack and programmer error.

Acknowledgements

The *Polar* project is a collaboration between research groups in the Department of Computing Science at the University of Newcastle, and the Department of Computer Science at the University of Manchester. We would like to thank Francisco Pereira and Jim Smith (Newcastle), and Norman Paton (Manchester) for discussions which have contributed to the contents of this paper. The *Polar* project is supported by the UK Engineering and Physical Science Research Council through grant GR/L89655.

References

1. Watson, P. and G.W. Catlow. *The Architecture of the ICL Goldrush MegaServer*. in *BNCOD13*. 1995. Manchester, LNCS 940, Springer-Verlag.
2. Loomis, M.E.S., *Object Databases, The Essentials*. 1995: Addison-Wesley.
3. Cattell, R.G.G., ed. *The Object Database Standard: ODMG 2.0*. , Morgan Kaufman.
4. Watson, P. and P. Townsend, *The EDS Parallel Relational Database System*, in *Parallel Database Systems*, P. America, Editor. 1991, LNCS 503, Springer-Verlag.
5. Watson, P. and E.H. Robinson, *The Hardware Architecture of the ICL Goldrush MegaServer*. Ingenuity- The ICL Technical Journal, 1995. **10**(2): p. 206-219.
6. Watson, P., M. Ward, and K. Hoyle. *The System Management of the ICL Goldrush Parallel Database Server*. in *HPCN Europe*. 1996: Springer-Verlag.
7. Bancilhon, F., C. Delobel, and P. Kanellakis, eds. *Building an Object-Oriented Database System : The Story of O2*. 1992, Morgan Kaufmann.
8. van den Berg, C.A., *Dynamic Query Processing in a Parallel Object-Oriented Database System*, 1994, CWI, Amsterdam.
9. Carey, M. et al., *Shoring up persistent applications*. in *1994 ACM SIGMOD Conf*. 1994. Mineapolis MN.
10. Day, M., et al., *References to Remote Mobile Objects in Thor*. ACM Letters on Programming Languages and Systems, 1994.
11. Rahm, E. *Dynamic Load Balancing in Parallel Database Systems*. in *EURO-PAR 96*. 1996. Lyon: Springer-Verlag.
12. Tamer Özsu, M. and P. Valduriez, *Distributed and parallel database systems*. ACM Computing Surveys,, 1996. **28**(1).
13. Hilditch, S. and C.M. Thomson, *Distributed Deadlock Detection: Algorithms and Proofs*, UMCS-89-6-1, 1989, Dept. of Computer Science, University of Manchester.
14. OMG, *The Common Object Request Broker: Architecture and Specification*. 1991: Object Management Group and X/Open.

15. Gerlhof, C.A., *et al.*, *Clustering in Object Bases*, TR 6/92. 1992, Fakultät für Informatik, Universität Karlsruhe.
16. Eickler, A., C.A. Gerlhof, and D. Kossmann. *A Performance Evaluation of OID Mapping Techniques*. in VLDB. 1995.
17. Gruber, O. and P. Valduriez, *Object management in parallel database servers*, in *Parallel Processing and Data Management*, P. Valduriez, Editor. 1992, Chapman & Hall. p. 275-291.
18. Singhal, V., S. Kakkad, and P. Wilson, *Texas: An Efficient, Portable Persistent Store*, in *Proc. 5th International Workshop on Persistent Object Stores*. 1992. p. 11-13.
19. Franklin, M.J., M.J. Carey, and M. Livny, *Transactional client-server cache consistency: alternatives and performance*. ACM Transactions on Database Systems, 1997. **22**(3): p. 315-363.
20. Sargeant, J., *Unified Functions and Objects: an Overview*, UMCS-93-1-4, 1993, University of Manchester, Department of Computer Science.
21. Adjeroh, D.A. and K.C. Nwosu, *Multimedia Database Management - Requirements and Issues*. IEEE Multimedia, 1997. **4**(3): p. 24-33.

Parallel Query Processing in a Shared-Nothing Object Database Server

L.A.V.C. Meyer M.L.Q. Mattoso

Computer Science Department
COPPE/UFRJ- Federal University of Rio de Janeiro
P.O. Box 68511, Rio de Janeiro, RJ, Brazil, 21945-970
e-mail: vivacqua, marta @cos.ufrj.br

Abstract. Parallel processing on OODBMS (Object Oriented Database Management Systems) may improve performance for non-conventional applications that manipulate large volumes of data. This work analyses and develops techniques that contribute for the improvement of query processing with shared-nothing (SN) parallel OODBMS. A solution for inter-node communication is developed where the effects of communication are minimised through the use of message queues, reduction of the message size and the creation of specific processes in each node.

1 Introduction

The object-oriented model is becoming very popular for database systems due to its structural and behavioural abstraction facilities. Data intensive applications with complex modelling requirements such as engineering, medicine and geography are natural candidates to the object oriented database systems (OODBMS). Those applications also impose high requirements for performance therefore parallel processing in OODBMS offer a promising solution to manage data in these new domains efficiently. Object-oriented query processing provides good opportunities to exploit parallel processing. However the richer modelling capabilities of the OODBMS impose some difficulties for an effective strategy for distributing objects across multiple disks. While in the relational model the database system operates with sets of tuples, in the OO model the database system has to operate with individual instances as well as with sets of objects. Also, the object clustering on disk is not uniform to the object class. Related sets of objects from different classes may be placed on the same disk page. Combining object-clustering algorithms with object partitioning is not a trivial task. Object partitioning is particularly important in shared-nothing (SN) database systems due to its static nature of object placement. SN database systems [10] with their promise of scalability and availability, have evolved as an answer to these new database applications. Therefore, a solution to parallel object database systems on shared nothing architectures have to deal with a good distributed design and special query processing techniques to reduce the communication overheads.

This work presents experimental object query processing results with the ParGoa parallel object server. We analyse and develop techniques that contribute for the improvement of query processing with shared-nothing (SN) systems. The ParGoa server is responsible for the parallel processing of the Goa++ OODBMS [9]. The Goa++ system is ODMG compliant and its ODL and OQL facilities are encapsulated with an ORB [15] (Object Request Broker) interface from the CORBA standard.

Experiments were made in a network of workstations configuring a SN parallel virtual machine with PVM software [8]. The performance of the prototype was evaluated in situations where there was no inter-node references and in situations where this kind of communication was necessary. Although parallel query has been explored extensively in the relational model, there are few experiments with parallel OO database systems or prototypes [2, 7, 13]. Shore [6] is an exception and this work adopts similar solutions to the Shore system. However while in [6] they analyse the effects of traversals without object transfer between nodes, this work focus on query processing with and without communication costs. The queries involve scanning the class extensions as well as traversals on different collections of objects. The goal of this paper therefore is to present a solution for inter-node object exchange where the effects of communication costs are minimised. The proposed solution involves the use of message queues, reduction of the message size and the creation of specific processes in each node for handling object transfers and local query processing.

This work is organised as follows. Section 2 introduces the Goa++ system while Section 3 presents the main features of the ParGoa server. Solutions for handling message passing in parallel object query processing are presented in Section 4. The object base where the experiments took place is addressed in Section 5. Performance results are shown and discussed in Section 6. Finally, Section 7 contains the conclusions.

2 The Goa++ System

The Goa++ object database system [9] works with a client/server architecture (Fig. 1). The client contains the application while the server executes the database persistence services and parallel query processing as well as other parallel set operations. The Goa++ client requests objects to the Goa++ server. The transfer unit between the client and server is the object (on collections) and not the page.

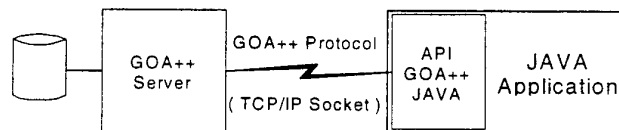


Fig. 1. GOA++ Server and JAVA Application

The Goa++ data model is compliant to ODMG standard therefore ODL is used to create a database schema and OQL is used to query stored collections of objects.

Goa++ is still a prototype and currently is single user. Therefore it lacks concurrency control. However, it presents some advanced features such as parallel processing, object distribution and metadata management.

The application programming interface (API) to Goa++ (Fig. 2) is a library of functions callable from C++ or Java applications. The dotted lines indicate modules under construction. OQL can be used as an embedded function in a programming language or as an "ad-hoc" query language.

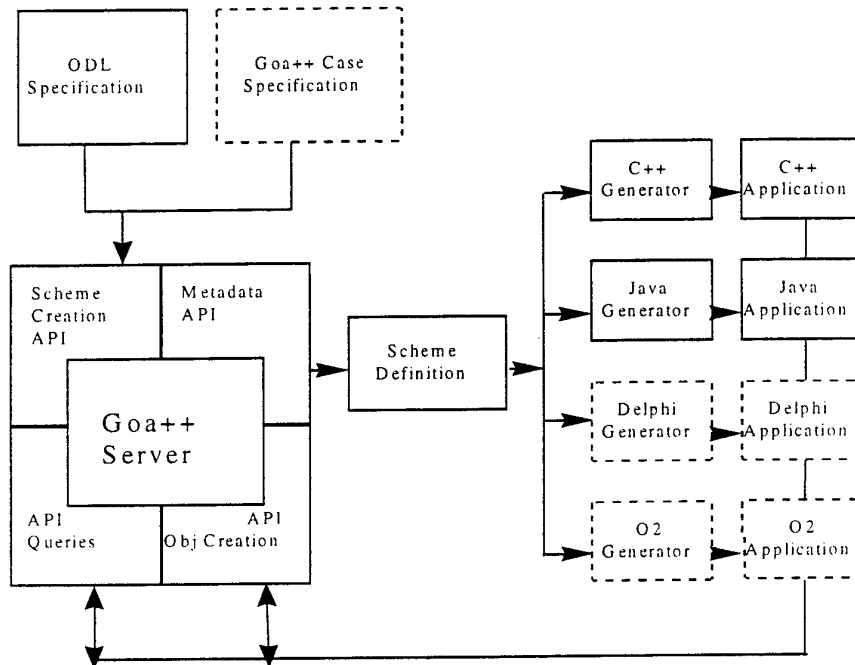


Fig. 2. Application programming interface to GOA

Goa++ Server has 4 levels of major classes that offer services of object management: ODMG compiler of languages, Schema Manager, Query Manager, and Object Storage Manager. We call the combination of Query and Object Storage Managers as the Goa Manager.

When Goa++ works with parallel query processing, the Goa Manager is replicated in several processing nodes. Also a new level of service is added above Goa Manager to co-ordinate the parallel execution. This server configuration is called ParGoa Server and is discussed in the next section.

3 The ParGoa Server

The ParGoa server is responsible for parallel processing on the Goa++ server. ParGoa services may be issued from the Goa++ client or different systems through the API or the ORB interface. Previous implementation of the ParGoa server explored the shared disk (SD) system. However, the shared disk access bottleneck led to the development of the SN parallel version ParGoa-SN. In the SN system the processors work with fewer disk data and may benefit from parallel disk access. Since the parallelism is achieved by the ParGoa server, this work presents the parallel query processing within the ParGoa-SN server (Fig 3). When a client submits a query, the ParGoa server executes sequentially the translation of the commands to an intermediary code. This code is analysed by the ParGoa Scheduler that generates the parallel execution plan of the query. After this phase, the query is executed in parallel without the interference of the Scheduler.

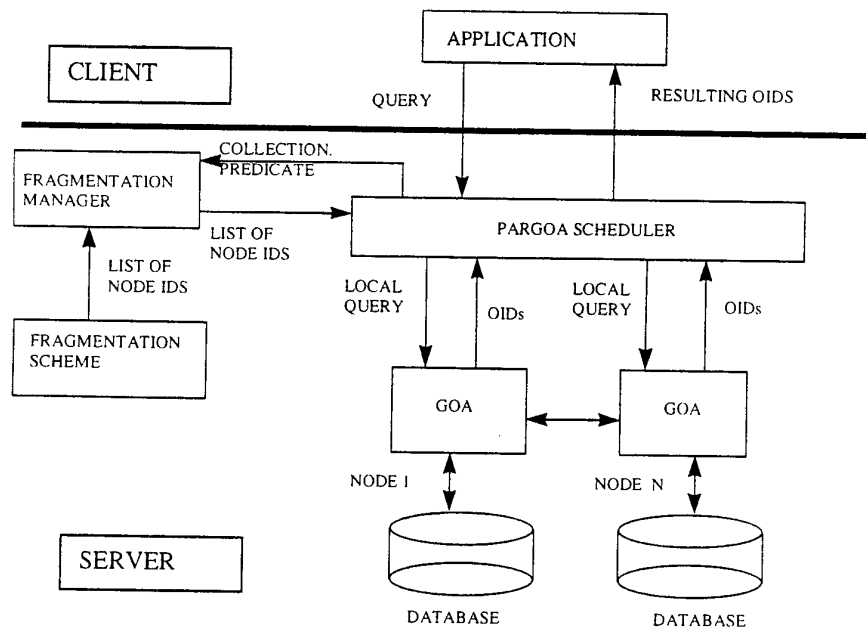


Fig. 3. The ParGoa-SN server

Once the schema is created, the database administrator may define a fragmentation policy to the ParGoa server. The Schema API does not handle this service because fragmentation is not supported by ODMG specification. Therefore

the fragmentation schema is directly sent to the ParGoa Scheduler. Therefore when the user submits an OQL query its execution is parallelized transparently by the ParGoa Scheduler. This version of ParGoa query processor does not create new objects or complex values as an answer as specified by ODMG. The parallel query always returns a set of object identifiers which may be requested and handled by the client.

The scheduler co-ordinates the execution of each ParGoa server operation, such as, queries, set operations and storage management. These operations are executed sequentially on each GOA node with the scheduler co-ordination. The GOA processing nodes are composed of a CPU, memory, and one disk drive. In this experiment the nodes use the Ethernet interconnection network for all communication. Each GOA node has a page cache of 4Mbytes.

The ParGoa-SN provides data parallelism by processing fragments of object sets in parallel. These fragments are managed by the Fragmentation Manager which allows primary and derived horizontal class fragmentation. The application can be given access to the entire distributed persistent object space since the ParGoa-SN provides location and fragmentation levels of transparency. The fragmentation and placement of objects during the database workload uses full declustering.

The ParGoa-SN provides data parallelism by processing fragments of object sets in parallel. These fragments are managed by the Fragmentation Manager which allows primary and derived horizontal class fragmentation. The application can be given access to the entire distributed persistent object space since the ParGoa-SN provides location and fragmentation levels of transparency. The fragmentation and placement of objects during the database workload uses full declustering.

3.1 ParGOA Scheduler

The ParGoa Scheduler is responsible for the allocation of the new objects and for the co-ordination of the parallel query processing executed on each node of the fragmented database. It communicates with the Fragmentation Manager and with the query processes that run on the Goa nodes. The Scheduler receives the query and sends it to the Fragmentation Manager. After receiving the identification of the nodes where the query must be processed, the local queries are sent to the specific Goa nodes where it is executed. It should be noted that the Scheduler and the Fragmentation Manager run on a single node, so a system bottleneck may occur only when the queries arrive at this node. After this, the parallel execution begins and there are no more interactions with the scheduler until each node finishes to execute its query.

3.2 Fragmentation Manager

The Fragmentation Manager is responsible for determining the nodes involved in the query execution. It uses the information stored at the fragmentation schema that describes how the classes are fragmented among the nodes. This schema has for each class, the type of its fragmentation (primary or derived), and the attribute with the corresponding ranges used to distribute the objects. To identify the nodes where the

ParGoa Scheduler must send the query, the Fragmentation Manager analyses the query predicate against the fragmentation schema.

4 Parallel Query Processing

When a query is submitted to ParGoa, the ParGoa Scheduler interacts with the Fragmentation Manager to see if the query may take advantage from the fragmentation strategy. When the fragmentation attribute is not involved in the query predicate, the Scheduler co-ordinates the query execution on all nodes, otherwise the query may be directed to specific fragments, i.e., nodes. The scheduler sends query predicates to be evaluated on each Goa node involved in the query execution. Query predicates may be classified in two categories: (1) *simple predicate* when the query involves only one class extension and (2) *complex predicate* when the query involves navigation (traversals) through different class objects. During the complex predicate evaluation a situation may occur where referenced objects might reside in a different node from the predicate evaluation node. To solve this kind of situation, two process were designed to execute in each node (Fig. 4).

The query process is responsible for the evaluation of the predicate and for the local query execution returning to Pargoa Scheduler the list of the objects that satisfy the operation. The object process caches and sends the required objects requested by other nodes. Since we used physical OID that aggregates the node identification, the query process knows for which fragment the message requesting the object needed in the predicate evaluation must be sent. Three approaches for inter-node communication were implemented. In the first one, the query process identifies that the object needed in the query predicate is placed in another node and immediately requests the object to be sent. This solution proved to be unsatisfactory, because of the large number of messages being exchanged among the nodes. To reduce the message traffic being exchanged between nodes, a second strategy was developed. The query process queues all the messages that request objects from other nodes until the query processing may no longer proceed. The object process receives a list of local object identifiers to be read from the disk and to be delivered to the requesting node.

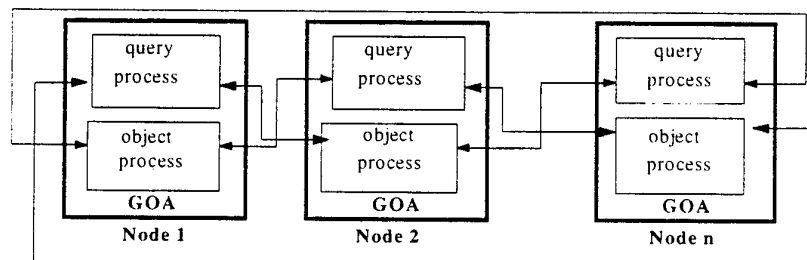


Fig. 4. Message handling within query processing

Although this second strategy led to significant performance improvement, a third solution was implemented. The object process instead of delivering all objects requested by other nodes, delivers only the attribute of the objects needed for the query evaluation, therefore reducing not only the number of messages exchanged among nodes but also the size of these messages.

5 The Partitioned Object Base

During the experiments the OO7 benchmark [1] was explored and this work presents just the execution over the interrelated classes shown in Figure 5. Few adaptations were made to the original data scheme because of some characteristics not supported by the system and out of the scope of the tests objectives.

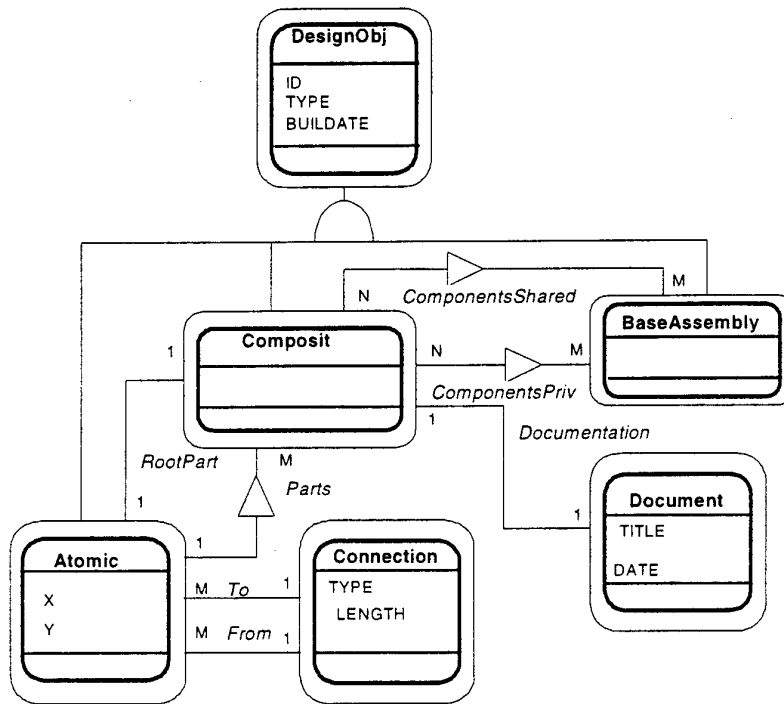


Fig. 5. OO7 adapted schema

The goal of the benchmark is to test many aspects of ODBMS performance, rather than to model a specific application. We used the number of objects as defined for the OO7 medium size database, so the number of composite parts was set to be 500 in our experiments. Each composite part is associated to a document object connected

by a bi-directional link and has an associated graph of atomic parts containing 200 objects. One atomic part in each composite part graph is designated the "root part". Each atomic part is connected via a bi-directional association to three other atomic parts. The connections between atomic parts are implemented by interposing a connection object between each pair of atomic parts. Figures 6 and 7 represent two different fragmentation strategies used on the evaluation of ParGoa. The first base partitioning uses primary fragmentation on Document class and all other classes in a derived way. Therefore the emphasis is in the relationship classes and all related objects are clustered in the same fragment. Thus in the partitioning strategy of Figure 6 there is no inter-node communication because the relationships are self-contained.

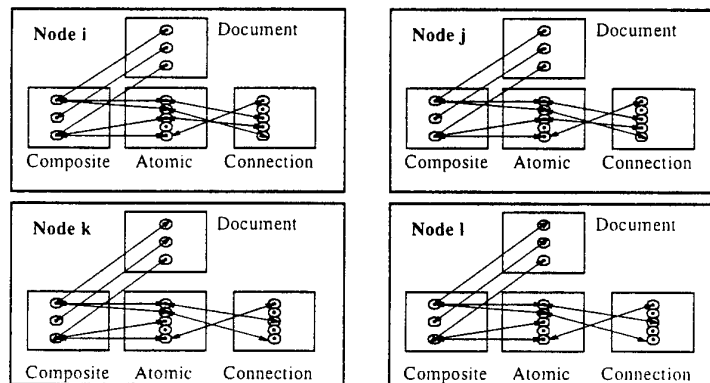


Fig. 6. Fragmentation without communication

However, since this is a special situation where objects are not shared between different nodes and load unbalance is likely to occur, another fragmentation policy was implemented.

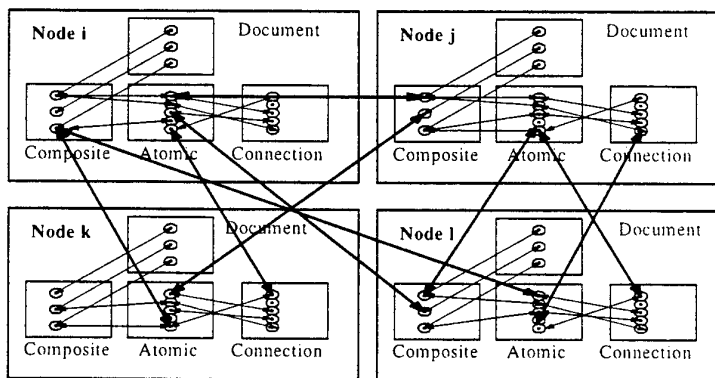


Fig. 7. Fragmentation with communication

In the second fragmentation strategy, the distribution of the objects among the nodes was changed forcing the necessity of communication between the nodes to follow the relationships as shown in Figure 7 by the darker lines. Although this situation is not favorable to SN database systems, it is more realistic than the former fragmentation. The goal of the fragmentation with communication was to measure the impact of this type of placement policy on traversals queries. The four classes were fragmented as follows: the Composite and Atomic classes were primarily fragmented on the date attribute while Document and Connection were derived to Composite and Atomic respectively. It should be noted that in the first fragmentation design the fragments might be processed independently from each other. However, while this situation is highly favourable to the SN model it may not be realistic without replication. Thus, Figure 7 represents a more realistic situation where the fragmentation and placement strategy enables direct execution to all the primary fragmented classes.

In each node, Goa uses collections to store the objects that will have associative access. A class of collection type, manages a list of the OIDs of the objects belonging to the collection. Objects can be clustered in the disk by object composition, object type or by object reference. Therefore a fragmented collection will have a subset of its OIDs in each node and the corresponding objects also stored in the local disk.

Four queries (Table 1) were evaluated on the experiments. Query 1 is part of the 007 specification, while Queries 2, 3 and 4 correspond to specified traversals embedded in queries. These queries represent a simple predicate in Query 1 and complex predicates in queries 2, 3 and 4.

Queries	Description
Query 1	Select Atomic where date < 1011.
Query 2	Select Atomic where Composite.Document.date < 1200
Query 3	Select Composite where Atomic.rootpart.date < 1011
Query 4	Select Connections where Atomic.date < 1011

Table 1. Implemented Queries

In Query 1 the Atomics may be scanned independently by all nodes with its own Atomic fragment. The simple queries represent an embarrassingly parallel operation. On the other hand, the complex predicates may involve crossing the boundary of the nodes and the two fragmentation strategies may behave differently. To take advantage of the distributed design, some queries coincide with the primary fragmentation and therefore may be directed to the specific nodes. The four queries are described as follows:

Query 1 \Rightarrow sequential scan query that access every Atomic selecting those satisfying a determined range.

Query 2 \Rightarrow navigational query traversing three classes selecting the Atomics that are part of a Composite whose document satisfies a determined range.

Query 3 \Rightarrow navigational query traversing two classes selecting the Composites having the Atomic root part satisfying a determined range.

Query 4 \Rightarrow navigational query traversing two classes with large cardinalities selecting the connections that connect atomic parts in the link having the date satisfying a determined range.

6 Performance Results

We implemented the ParGoa-SN and performed our tests on a cluster of IBM RS/6000(Powerpc) stations connected by Ethernet. Each workstation had 32MB of main memory. The IBM Stations in the cluster were not isolated, and since we did not have exclusive access to these workstations we did not kill the usual suite of daemons and background processes. However, we did ensure that there were no active users on the workstations when the tests were run.

The time measured was the elapsed (wall clock) time which is the time that the user waits for the problem solution. This kind of measure accounts not only for the computational work, but also for any waiting for locks, paging and I/O [4]. The results show performance speedup for situations (hot) where the cache was not empty. To reduce the interference effects due to not having isolated workstations, we re-ran each query 20 times.

We start to measure the execution time when the ParGoa Scheduler sends the query to all the slave nodes that are involved in the query until the Scheduler receives results from all those slave nodes. It should be noted that the results showed low variance and the elapsed time for each query is presented in [11].

These results reflect the two fragmentation strategies presented in Figures 6 and 7 with the medium size OO7 database. We begin by showing the query results for the fragmentation that involves no communication (Fig. 6) and then we present the corresponding results for the second strategy.

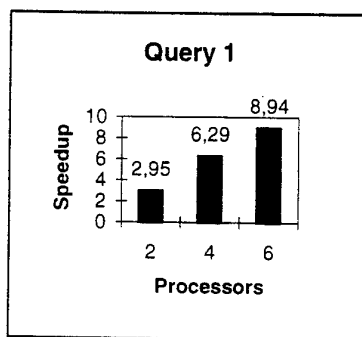


Fig. 8. Query 1 with no comm.

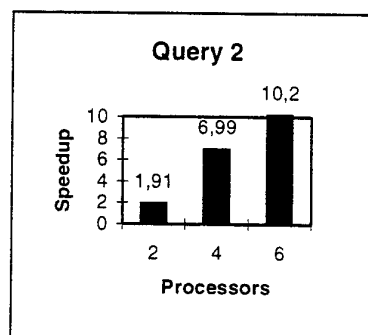


Fig. 9. Query 2 with no comm.

The results for Queries 1 and 2 in the first fragmentation design are presented in Figures 8 and 9. These results show significant speedup as expected. The super-linear speedup corresponds to the additional memory obtained with the additional nodes. However, since the cache was limited to 4 Mbytes, even with six nodes, the accessed objects could not fit in the available memory. The Query 2 execution showed better results because the query could be directed to execute in only one node.

Figures 10 and 11 show the results for Queries 3 and 4. As mentioned before, Query 3 selects the composite parts that present the root part satisfying a condition, while Query 4 selects the connections that connect atomic parts whose date satisfy a condition.

Both of these navigational queries were executed on all nodes of the parallel environment. Although presenting a good performance during parallel executions, Query 3 did not show a linear speedup because of the random access performed to the root-part. Query 4 presented better results as more processors were added to the experiment.

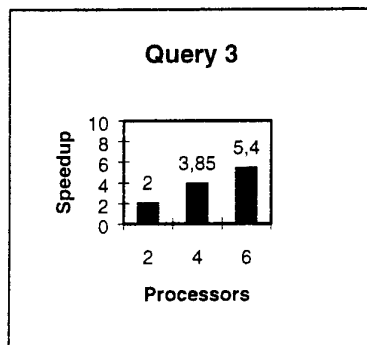


Fig. 10. Query 3 with no comm.

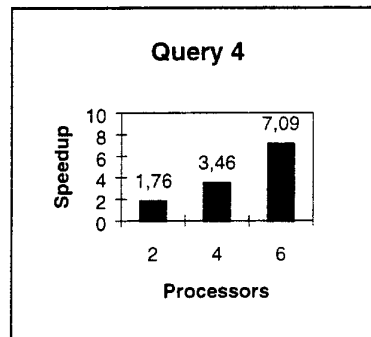


Fig. 11. Query 4 with no comm.

Figures 12 and 13 present the results for the Queries 1 and 2 but with the second fragmentation design which involves communication to follow the inter-node object references shown in Figure 7. The execution of Query 1 presented the same results from the situation without communication. Since this query accesses only one class extension, no communication was necessary in either situation.

The execution of Query 2 involved all nodes and communication between them to follow the object references. However, the performance results are still around linear speedup. This efficiency was a result from the two process architecture that diminished the communication traffic.

This individual instance access typical of object oriented systems imposed more traffic on the communication network. Previous experiments of these same Queries with object communication and without the message queue and the object process

presented no speed up at all. Thus showing the impact of communication in object oriented environments.

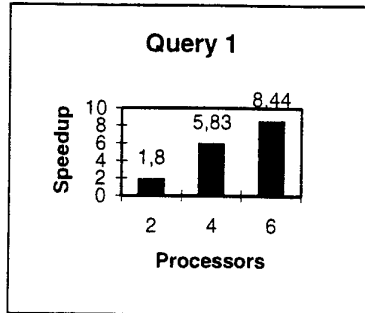


Fig. 12. Query 1 with comm.

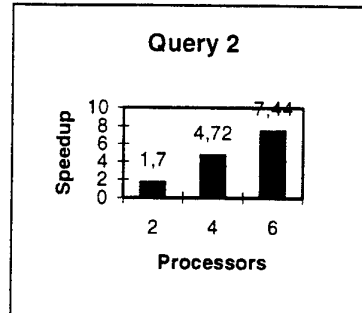


Fig. 13. Query 2 with comm.

Query 3 (Fig. 14) was also executed in all nodes of the system in the second fragmentation design. However, this second placement strategy forces the nodes to exchange messages during the navigational query. Query 4 (Fig. 15) according to this new design situation has the execution directed to only one node. As can be observed in Figure 14, Query 3 presented better results in this situation than in the first one. Although involving all nodes and communication between them to follow the object references in the navigational predicate, in this strategy the query processor only follows the references after scanning the whole local collection. Therefore it deals with the different collections separately through its object process. Re-running Query 3 with the strategy of having two processes in each node, but with the no communication design, the speedup obtained was higher than the situation with fragmentation with communication.

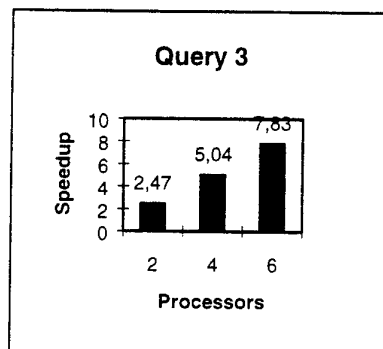


Fig. 14. Query 3 with comm.

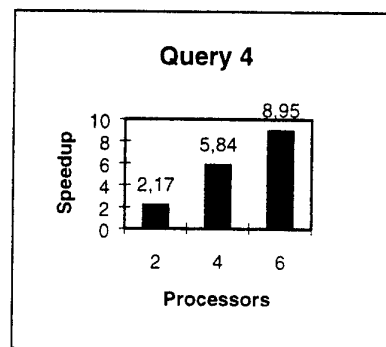


Fig. 15. Query 4 with comm.

The results for Query 4 (Fig. 15) also showed better speedup than in the first situation since in this second placement strategy, the query execution was done in only one node. This same behavior was presented by Query 2, where in the first design situation, the fragmentation manager could take advantage from the fragmentation strategy and direct the query to be executed in only one node. These results make evident the impact of the distribution design in the performance of navigational queries in SN parallel OODBMS.

7 Conclusion

Parallel processing on OODBMS may improve performance for non-conventional applications that manipulate large volumes of data. This work analyses and develops techniques which contribute for the improvement of query processing with shared-nothing (SN) parallel OODBMS. A strategy to solve external references for the evaluation of query predicates was implemented. The experiments were made in a network of workstations configuring a parallel virtual machine with PVM software. The performance of the prototype was evaluated in situations where there was no inter-node references and in situations where this kind of communication was necessary. The application used in the experiments presented potential speedup even in situations where the communication was intensive. This work presents techniques for inter-node communication where the effects of communication are minimised through the use of message queues, reduction of the message size and the creation of specific processes in each node. The results also stressed the benefits from the primary and derived fragmentation strategy for distributing objects. Finally, the performance results in this work show how performance may improve and be handled in shared nothing systems with object orientation.

References

1. CAREY,J., et al., 1993, "The 007 Benchmark", In: Proceedings of ACM SIGMOD International Conference on Management of Data, Washington DC, U.S.A., pp.12-21.
2. CHEN,YAW., SU,STANLEY., 1996, "Implementation and Evaluation of Parallel Query Processing Algorithms and Data Partitioning Heuristics in Object-Oriented Databases", Distributed and Parallel Databases, vol. 4(2), pp. 107-141.
3. COOK, J., et al., 1996, "Semi-Automatic, Self-Adaptative Control of Garbage Collection Rates in Object Databases", In: Proceedings of ACM SIGMOD International Conference on Management of Data, Montreal, Canada, pp. 377-388.
4. CROWL,L., 1994, "How to Measure, Present and Compare Parallel Performance", IEEE Parallel and Distributed Technology, vol. 2(1), pp.9-25.
5. DEWITT,D.,GRAY,J., 1992, "Parallel Database Systems: The Future of High Performance Database Systems", Communications of the ACM, vol. 35(6), pp.85-98.
6. DEWITT,D., et al., 1996, "Parallelizing OODBMS Traversals: a performance evaluation" VLDB Journal vol. 5(1), pp. 3-18.

7. GHANDEHARIZADEH,S., et al., 1994, "Object Placement in Parallel Object-Oriented Database Systems", In: Proceedings of the 10th International Conference on Data Engineering, Houston, USA, pp. 253-262.
8. GEIST,A., et al., 1994, PVM 3 USER'S GUIDE AND REFERENCE MANUAL, ORNL/TM-12187, Oak Ridge National Laboratory, Tennessee, USA.
9. MAURO,R. et al., 1997, "GOA++: Technology, Implementation and Extensions to the Object Management Services", In: Proceedings of the 12th Brazilian Conference on Databases, Fortaleza, Brazil, pp.272-286. (URL-<http://www.cos.ufrj.br/~goa>)
10. METHA, M., DEWITT,D., 1997, "Data Placement in Shared-nothing Parallel Database Systems" VLDB Journal vol. 6(1), pp. 53-72.
11. MEYER, L.A.V.C., 1997, "Shared-nothing Parallelism in OODBMS: an Implementation with ParGOA", (in portuguese) M.Sc. Thesis, COPPE/UFRJ, Rio de Janeiro, RJ, Brazil.
12. NORMAN,M. ZUREK,T. THANISCH,P., 1996, "Much Ado About Shared-Nothing", SIGMOD RECORD, vol. 35(3), pp. 16-21.
13. THAKORE,A. SU,S., 1994, "Performance Analysis of Parallel Object-Oriented Query Processing Algorithms", Distributed and Parallel Databases, vol. 2(1), pp. 59-99.
14. VALDURIEZ,P., 1993, "Parallel Database Systems:open problems and new issues", Distributed and Parallel Databases,vol.1(2), pp. 137-165.

High Performance Computing of a New Numerical Algorithm for an Industrial Problem in Tribology *

M. Arenaz¹, R. Doallo¹, G. García², and C. Vázquez³

¹ Department of Electronics and Systems, University A Coruña,
15071 A Coruña, Spain
arenaz@des.fi.udc.es , doallo@udc.es

² Department of Applied Mathematics, University of Vigo, 36280 Vigo, Spain
guille@dma.uvigo.es

³ Department of Mathematics, University A Coruña, 15071 A Coruña, Spain
carlosv@udc.es

Abstract. In this work we present the vectorization of a new complex numerical algorithm to simulate the lubricant behaviour in an industrial device issued from tribology. This real technological problem leads to the mathematical model of the thin film displacement of a fluid between a rigid plane and an elastic and loaded sphere. The mathematical study and a numerical algorithm to solve the model has been proposed in the previous work [9]. This numerical algorithm mainly combines fixed point techniques, finite elements and duality methods. Nevertheless, in order to obtain a more accurate approach of different real magnitudes, it is interesting to be able to handle finer meshes which increase storage and computation costs. So, in order to increase the performance of the numerical algorithm in terms of execution time, in this work we mainly apply vectorization techniques and present some preliminary partial results from the design of a parallel version of the algorithm. Several test examples corresponding to different real data sets are presented to illustrate the advantages of high performance computing.

1 The Industrial Problem in Tribology

In a wide range of lubricated industrial devices studied in Tribology the main task is the determination of the fluid pressure distribution and the gap between the elastic surfaces which correspond to a given imposed load [4]. Most of these devices can be represented by a ball-plane geometry (see Fig. 1). So, in order to perform a realistic numerical simulation of the device, an appropriate mathematical model must be considered. From the mathematical point of view, the lubricant pressure is governed by the well-known Reynolds equation [4]. In the

* Partially supported by Research Projects of Xunta de Galicia (XUGA 20605B96 and XUGA 32201B97), Ministerio de Educación y Ciencia of Spain (CICYT TIC 96-1125-C03) and D.G.E.S. (PB96-0341-C02).

case of elastic surfaces, the computation of the lubricant pressure is coupled with the determination of the gap. Thus, in Reynolds equation the gap depends on the pressure. In the particular ball-bearing geometry, the local contact aspect allows to introduce the Hertz contact theory to express this gap-pressure dependence. The inclusion of cavitation (the presence of air bubbles) and piezoviscosity (pressure-viscosity dependence) phenomena is modelled by a more complex set of equations, see [5]. Moreover, the balance between imposed and hydrodynamic loads is formulated as a nonlocal constraint on the fluid pressure, see [9] for the details about the mathematical formulation.

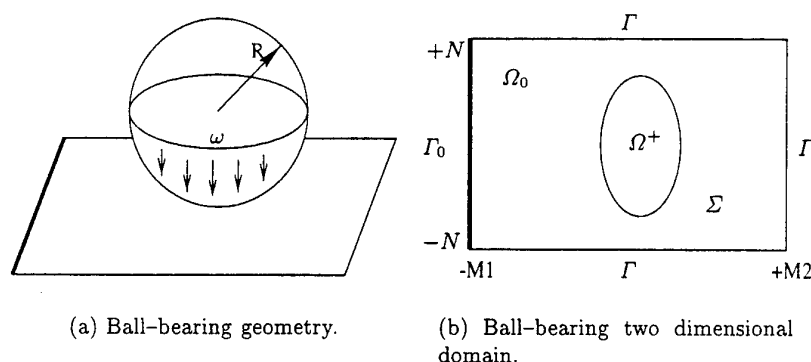


Fig. 1. Ball-bearing device.

In Sections 2 and 3 we briefly describe the model problem and the numerical algorithm presented in [9], respectively. In Section 4 we make reference to the vectorization and parallelization techniques that we have applied in order to improve the performance of the algorithm. In Section 5 we present the numerical results and the execution times corresponding to several different meshes. Finally, in Section 6 we present the conclusions we have come to as a result of our work.

2 The Model Problem

As a previous step to describe the numerical algorithm, we introduce the notations and the equations of the mathematical model.

Thus, let Ω be given by $\Omega = (-M_1, M_2) \times (-N, N)$, with M_1, M_2, N positive constants, which represents a small neighbourhood of the *contact point*. Let $\partial\Omega$ be divided in two parts: the supply boundary $\Gamma_0 = \{(x, y) \in \partial\Omega / x = -M_1\}$ and the boundary at atmospheric pressure $\Gamma = \partial\Omega \setminus \Gamma_0$.

In order to consider the cavitation phenomenon, we introduce a new unknown θ which represents the saturation of fluid ($\theta = 1$ in the fluid region where $p > 0$ and $0 \leq \theta < 1$ in the cavitation region where $p = 0$). The mathematical

formulation of the problem consists of the set of nonlinear partial differential equations (see [2] and [6] for details) verified by (p, θ) :

$$\frac{\partial}{\partial x} \left(\frac{\rho}{\nu} h^3 \frac{\partial p}{\partial x} \right) + \frac{\partial}{\partial y} \left(\frac{\rho}{\nu} h^3 \frac{\partial p}{\partial y} \right) = 12s \frac{\partial}{\partial x} (\rho h), \quad p > 0, \quad \theta = 1 \quad \text{in } \Omega^+ \quad (1)$$

$$\frac{\partial}{\partial x} (\rho \theta h) = 0, \quad p = 0, \quad 0 \leq \theta \leq 1 \quad \text{in } \Omega_0 \quad (2)$$

$$\frac{h^3}{\nu} \frac{\partial p}{\partial n} = 12s(1 - \theta) h \cos(\mathbf{n}, \mathbf{z}), \quad p = 0 \quad \text{in } \Sigma \quad (3)$$

where the lubricated region, the cavitated region and the free boundary are

$$\Omega^+ = \{(x, y) \in \Omega / p(x, y) > 0\}$$

$$\Omega_0 = \{(x, y) \in \Omega / p(x, y) = 0\}$$

$$\Sigma = \partial\Omega^+ \cap \Omega$$

and where p is the pressure, h the gap, $(s, 0)$ the velocity field, ν the viscosity, ρ the density, \mathbf{n} the unit normal vector to Σ pointing to Ω_0 and \mathbf{z} the unit vector in the x -direction.

In the elastic regime the gap between the sphere and the plane is governed by (see [10] and [6]):

$$h = h(x, y, p) = h_0 + \frac{x^2 + y^2}{2R} + \frac{2}{\pi E} \int_{\Omega} \frac{p(t, u)}{\sqrt{(x-t)^2 + (y-u)^2}} dt du \quad (4)$$

where h_0 is the minimum reference gap, E is the Young equivalent modulus and R is the sphere radius. The Equation (4) is issued from hertzian contact theory for local contacts. The relation between pressure and viscosity is:

$$\nu(p) = \nu_0 e^{\alpha p} \quad (5)$$

where α and ν_0 denote the piezoviscosity constant and the zero pressure viscosity, respectively. Moreover, the boundary conditions are:

$$\theta = \theta_0 \quad \text{in } \Gamma_0 \quad (6)$$

$$p = 0 \quad \text{in } \Gamma \quad (7)$$

The above conditions correspond to a drep feed device where the lubricant is supplied from the boundary Γ_0 . Finally, the hydrodynamic load generated by fluid pressure must balance the load ω , imposed on the device, in a normal direction to the plane (see Fig. 1(a)):

$$\omega = \int_{\Omega} p(x, y) dx dy \quad (8)$$

In this model, the parameter h_0 appearing in Equation (4) is an unknown of the problem related to this condition. The numerical solution of the problem consists of the approximation of (p, θ, h, h_0) verifying (1)-(8).

3 The Numerical Algorithm

In order to perform the numerical solution of (1)-(8), with real industrial data, we previously proceed to its adimensionalization in terms of the load, radius and material data. This adimensionalization leads to a fixed domain $\Omega = (-4, 2) \times (-2, 2)$ by introducing the hertzian contact radius and the maximum hertzian pressure,

$$b = \left(\frac{3\omega R}{2E} \right)^{\frac{1}{3}}, \quad P_h = \frac{3\omega}{2\pi b^2},$$

respectively, which only depend on the imposed load ω , the Young modulus E and the sphere radius R . Thus, the new dimensionless variables to be considered are:

$$\tilde{p} = \frac{p}{P_h}, \quad \tilde{h} = \frac{hR}{b^2}, \quad X = \frac{x}{b}, \quad Y = \frac{y}{b},$$

$$\bar{\alpha} = \alpha P_h, \quad \bar{\nu} = \frac{\nu}{\nu_0}, \quad \lambda = \frac{8\pi\nu_0 s R^2}{\omega b}, \quad \bar{\omega} = \frac{\omega}{P_h b^2} = \frac{2\pi}{3}.$$

After this adimensionalization, the substitution in (1)-(8) leads to equations of the same type in the dimensionless variables. In [9] a new numerical algorithm is proposed to solve this new set of equations. In this paper, high performance computing techniques are applied to this algorithm, which is briefly described hereafter.

The first idea is to compute the hydrodynamic load (Eqn. (8)) for different gap parameters \tilde{h}_0 in order to state a monotonicity between \tilde{h}_0 and this hydrodynamic load. Then, the numerical solution of the problem for each \tilde{h}_0 is decomposed in the numerical solution of:

1. The hydrodynamic problem: Computation of the pressure and saturation for a given gap. A previous reduction to an isoviscous case is performed in order to vanish the nonlinearity introduced by (5). In this step, the characteristics algorithm, finite elements and duality methods are the main tools.
2. The elastic problem: Computation of the gap for a given pressure by means of numerical quadrature formulae to approximate the expression (4). This computation can be expressed by means of the loop shown below:

LOOP in n (Gap Loop)

$$h(p^{n+1}) = h_0 + \frac{x^2 + y^2}{\pi E} \sum_{k \in \tau_h} \int_k \frac{p^{n+1}(t, u)}{\sqrt{(x-t)^2 + (y-u)^2}} dt du \quad (9)$$

In the previous formula the updating of the gap requires the sum of the integrals over each triangle k of the finite element mesh τ_h .

The complex expression of the gap at each mesh point motivates the high computational cost of this step of the algorithm to obtain $h(p^{n+1})$.

An outer loop for the gap and pressure computations for each value of h_0 determines the value of h_0 which balances the imposed load (Eqn. (8)). This design of the algorithm is based on monotonicity arguments and *regula falsi* convergence. In a first term, an interval $[h_a, h_b]$ which contains the final solution \bar{h}_0 is obtained. Then, this value is computed by using a *regula falsi* method.

In order to clarify the structure of the code, we present a flowchart diagram of the numerical algorithm in Fig. 2.

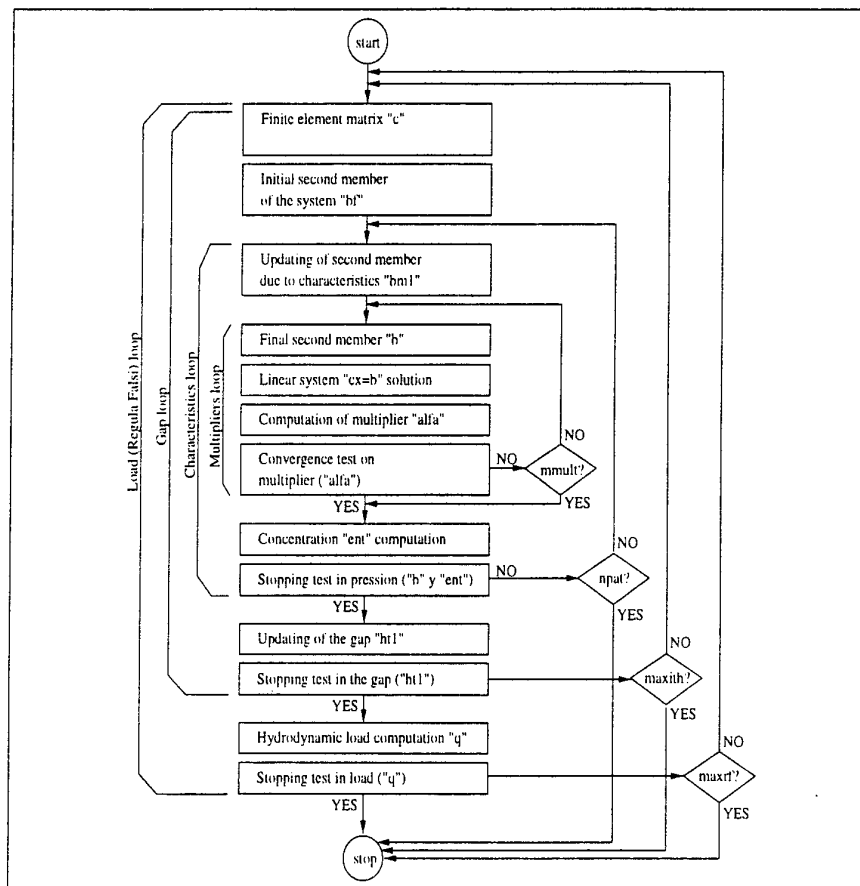


Fig. 2. Flowchart diagram of the algorithm in terms of functional blocks. Next to the identifier of each functional block appears, in inverted commas, the name of the programme variable updated as a result of the calculus. The parameters *mmult*, *npat*, *maxith* and *maxrf* represent the maximum number of iterations for multipliers, characteristics, gap and load loops, respectively.

A numerical flux conservation test shows that better results correspond to finer grids. Nevertheless, this mesh refinement greatly increases the computing times and motivates the interest of using high performance computing.

4 Improving Algorithm Performance

The main goal of our work is to increase the performance of the new numerical algorithm [9] briefly explained above. In order to carry out our work, we have considered two different approaches: in first place, the use of vectorization techniques and, in second place, the use of parallelization techniques.

The first part of our work consisted on modifying the original source code of the programme in order to get the maximum possible performance from a vector computer architecture. The target machine was the Fujitsu VP2400/10 available at the CESGA laboratories¹.

The second step of our work consisted on developing a parallel version of the original sequential source code by using the PVM message-passing libraries. In order to check the performance of the parallelization process, we have executed our parallel programme over a cluster of workstations based on a SPARC micro-processor architecture at 85 MHz and Solaris operating system interconnected via an Ethernet local area network.

4.1 Vectorization Techniques

The first step that must be done when trying to vectorize a sequential programme is to analyze its source code and identify the most costly parts of the algorithm in terms of execution time. We will focus most of our efforts on these parts. We used the *ANALYZER* tool [7] in order to carry out this task.

The analysis of the programme's source code lead us to identify nine functional blocks according to their relationship with the mathematical methods used in the numerical algorithm, namely, finite elements, characteristics and duality methods. These blocks, shown in Fig. 2, are the following: finite element matrix, initial second member of the system, updating of the second member due to characteristics, final second member, computation of the multiplier, convergence test on multipliers, stopping test in pressure, updating of the gap and stopping test in the gap.

We have done a study of the distribution of the total execution time of the programme, and we have come to the conclusion that the 90% of that time is taken by the blocks included in the multipliers loop (final second member and computation of the multiplier) and by the block that updates the gap. For this reason, in this paper we explain the vectorization process corresponding to the three blocks that we have just mentioned. More detailed information about the whole vectorization process can be found in [1].

¹ CESGA (Centro de Supercomputación de Galicia): Supercomputing Center of Galicia placed at Santiago de Compostela (Spain).

Final Second Member. The first block inside the multipliers loop computes the final value of the second member of the linear equations system that is solved in the current iteration of the loop.

This block is divided in two parts, the first one is only partially vectorizable by the compiler because of the presence of a recursive reference. Table 1 shows the execution times obtained for this part before and after the vectorization process according to the format $h : m : s.\mu s$, where h , m , s and μs represent the number of hours, minutes, seconds and microseconds, respectively.

Table 1. Execution times corresponding to the routine *modsm()*.

	<i>T_Orig</i>	<i>T_AV</i>	<i>iprv_AV</i>	<i>T_MV</i>	<i>iprv_MV</i>
<i>mesh1</i>	0.001214	0.000526	56.67%	0.000325	73.23%
<i>mesh3</i>	0.015122	0.005188	65.69%	0.004595	69.61%
<i>mesh4</i>	0.060288	0.020446	66.09%	0.018160	69.88%
<i>mesh5</i>	0.120261	0.040887	66.00%	0.036354	69.77%

The first column points out which mesh has been used for the discretization of the domain of the problem, where *mesh1*, *mesh3*, *mesh4* and *mesh5* consist of 768, 9600, 38400 and 76800 finite elements, respectively. The column labeled as *T_Orig* shows the time measured when executing the programme after a scalar compilation. The columns labeled as *T_AV* and *T_MV* show, respectively, the times after setting the automatic vectorization compiler option and after applying this last option over the modified source code we have implemented in order to optimize the vectorization process. The columns *iprv_AV* and *iprv_MV* reveal the improvement with regard to the original scalar execution time. The percentages shown have been calculated by means of the following expressions:

$$iprv_AV = 100 * (1 - (T_AV/T_Orig)) \quad (10)$$

$$iprv_MV = 100 * (1 - (T_MV/T_Orig)) \quad (11)$$

The second part of this block has been vectorized by making use, on the one hand, of the loop coalescing technique [11] and, on the other, of the *in-lining* transformation technique. The last one has been applied manually so as to be able to optimize the source code of the particular case we are interested in. This resource introduces a generality loss in the vector version of the programme with regard to its original source code. We consider this fact acceptable because our goal was to reduce the execution time as much as possible.

Table 2 shows the execution times corresponding to this fragment of code. It is important to highlight the high improvement we have obtained with respect to the simple automatic vectorization.

Computation of the Multiplier. The third block inside the multipliers loop mainly copes with the updating of the multiplier introduced by the duality type

Table 2. Execution times corresponding to the routine *bglfsm()*.

	<i>T_Orig</i>	<i>T_AV</i>	<i>iprv_AV</i>	<i>T_MV</i>	<i>iprv_MV</i>
<i>mesh1</i>	0.001231	0.001255	-1.95%	0.000508	58.73%
<i>mesh3</i>	0.015383	0.015339	0.29%	0.005918	61.53%
<i>mesh4</i>	0.059497	0.057986	2.54%	0.023546	60.42%
<i>mesh5</i>	0.118632	0.115769	2.41%	0.047059	60.33%

method proposed in [3]. Besides, it makes the necessary computations so as to implement the convergence test on multipliers.

In the first case, we are faced with the fact that there was a loop that was not automatically vectorizable by the compiler because of the presence of, on the one hand, a call to an external function and, on the other, a recursive reference on the vector whose value is computed, namely, *alfa*.

The first of these problems was solved by applying manually the *in-lining* technique. Again the resultant source code was optimized for the particular case we are interested in, with the corresponding loss of generality.

The recursive reference present in the source code is not vectorizable because when computing the *i*-th element of the vector *alfa*, the (*i* - *k*)-th element is referenced, being *k* a positive constant with a different value for each mesh. The use of the compiler directive **VOCL LOOP,NOVREC(alfa)* (see [8]), which explicitly tells the compiler to vectorize the recursive reference, has allowed us to overcome the problem. We have checked that the numerical results keep on being the correct ones, improving the percentage of vectorization of our code.

Table 3 shows the execution times, expressed in microseconds, corresponding to the process described in the previous paragraphs.

Table 3. Execution times corresponding to the updating of the multiplier.

	<i>T_Orig</i>	<i>T_AV</i>	<i>iprv_AV</i>	<i>T_MV</i>	<i>iprv_MV</i>
<i>mesh1</i>	0.001115	0.001113	0.18%	0.000043	96.14%
<i>mesh3</i>	0.012785	0.012903	-0.92%	0.000162	98.73%
<i>mesh4</i>	0.050319	0.049284	2.06%	0.000489	99.03%
<i>mesh5</i>	0.100197	0.098290	1.90%	0.000930	99.07%

As can be seen, improvement percentages are close to 100% (moreover, see the great improvement obtained with respect to the one obtained enabling automatic vectorization without providing any extra information to the compiler). If we bear into account that this functional block is placed in the body of the innermost loop of the algorithm (multipliers loop), which is the section of the programme that is executed the greatest number of times, we can conclude that the reduction of the execution time of the whole programme will be very important.

Updating of the Gap. The third and last block whose vectorization process is presented in this paper copes with the computation of the gap between the two surfaces in contact in the ball-bearing geometry of Fig. 1(a). The source code corresponding to this block was fully vectorizable by the compiler. Nevertheless, we have carried out some changes by using source code optimization techniques and the use of scalar variables in reduction operations technique.

Table 4 shows execution times corresponding to the updating of the gap.

Table 4. Execution times corresponding to the updating of the gap.

	<i>T_Orig</i>	<i>T_AV</i>	<i>iprv_AV</i>	<i>T_MV</i>	<i>iprv_MV</i>
<i>mesh1</i>	0:00:01.351421	0:00:00.060134	95.55%	0:00:00.052062	96.15%
<i>mesh3</i>	0:03:28.791518	0:00:08.546158	95.91%	0:00:07.511769	96.20%
<i>mesh4</i>	0:51:53.848528	0:02:17.901371	95.57%	0:02:01.627059	96.09%
<i>mesh5</i>	3:26:54.230843	0:09:09.402314	95.57%	0:08:05.177401	96.09%

As can be seen, execution time improvement percentage is slightly higher than 96% independently of the size of the mesh.

Although this part of the algorithm is executed a relatively low number of times, it is by large the most costly of the programme (approximately, 14% of the total scalar execution time). This is due to the fact that for each node of the mesh (the number of nodes is 19521 in *mesh4*, for example) it is necessary to compute an integral over all the domain of the problem. That integral is decomposed in a sum of integrals over each finite element (the number of elements is 38400 in *mesh4*) which are solved with numerical integration.

In Section 5 we present the times corresponding to the execution of the whole programme.

4.2 Parallelization Techniques

The parallel algorithm version is at this moment being refined. In this paper we will focus our attention on the block corresponding to the updating of the gap between the two surfaces in lubricated contact (see corresponding part of Subsection 4.1).

For each node of the mesh, the updating of the gap is done according to (9). If we analyze the source code corresponding to this computation, we can see that we are faced with two nested loops where the outermost makes one iteration for each node of the mesh and the innermost makes one iteration for each finite element.

Each iteration of the outer loop calculates the gap corresponding to one node of the mesh. This computation is cross-iteration independent. This characteristic makes the efficiency of the process of parallelization to be independent of the data distribution among the workstations, which lead us to choose one simple data distribution such as the standard consecutive distribution.

Table 5 shows the execution times obtained for this functional block, after the parallelization process. Fig. 3 represents, for each mesh, the *speedups* corresponding to the execution times shown in Table 5.

Table 5. Execution times corresponding to the updating of the gap.

N	mesh1	mesh3	mesh4	mesh5
1	3.961	9:29.688	2:29:35.406	9:52:36.750
2	1.872	4:21.219	1:08:52.500	
4	0.953	2:19.834	0:34:35.219	
8	0.598	2:27.909	0:17:24.781	1:08:56.719
12	0.542	1:18.613	0:11:40.469	0:46:11.344
16	0.577	1:00.074	0:08:54.438	0:34:57.156

The *speedup* is usually defined according to the following formula:

$$speedup = T_1/T_N \quad (12)$$

where T_N represents the time when using N processors and T_1 the time obtained when executing the parallel programme over only one processor. In almost all cases it is obtained a *superspeedup*, that is to say, the value of the *speedup* is greater than the number of processors used. This effect may be due to the fact that we have distributed the vector of the gaps corresponding to each node of the mesh among the set of workstations. In consequence, we have reduced the size of that vector, which has decreased the number of cache misses.

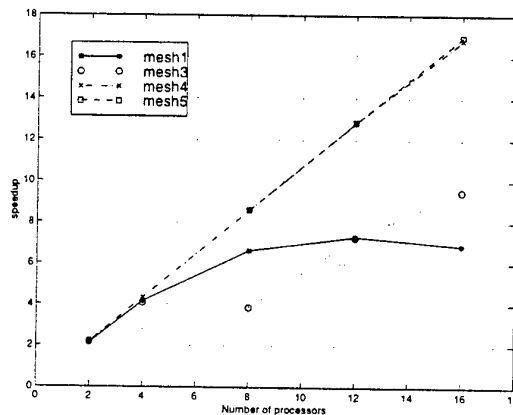


Fig. 3. Speedups corresponding to data shown in Table 5.

Note also that the performance of the programme decreases when using small meshes (*mesh1* and *mesh3*) with a high number of workstations. This effect is due to the communications overhead.

5 Numerical Results

The results obtained from the vectorized code were validated by comparing them with those ones calculated with the execution of the scalar code.

In Table 6 different execution time measures illustrate the good performance achieved after applying code and compiler optimization techniques with respect to the one obtained by the vectorizer without any extra code information. The table also shows the scalar execution time. In these cases the coarser grid, namely *mesh1* (768 triangular finite elements), is used for the spatial discretization.

Table 6. Execution times for *mesh1* on the VP2400/10.

	T_Orig	T_AV			
		TVU	TCPU	% Vect.Exec.	% Vectorization
<i>b</i>	11:24	01:45	07:03	24.82%	53.51%
<i>l</i>	05:03	00:42	02:47	25.15%	58.75%
<i>g</i>	15:14	02:38	10:53	24.20%	45.84%
<i>j</i>	07:32	01:14	05:03	24.42%	49.34%

	T_Orig	T_MV			
		TVU	TCPU	% Vect.Exec.	% Vectorization
<i>b</i>	11:24	02:31	03:08	80.32%	94.59%
<i>l</i>	05:03	01:00	01:16	78.95%	94.72%
<i>g</i>	15:14	03:50	04:47	80.14%	94.76%
<i>j</i>	07:32	01:48	02:14	80.60%	94.25%

The first column presents the different input data sets, labeled as *b*, *l*, *g* and *j*, that have been employed for the tests. The column *T_Orig* shows the scalar execution time of the corresponding tests and the group of three columns labeled as *T_AV*, the times corresponding to the execution of the automatically vectorized code. The column *TCPU* represents the total execution time. The column *TVU* refers to the programme execution time on the vector units of the VP2400/10. The column *%Vect.Exec.* (vector execution percentage) measures the percentage of the execution time that is carried out on the vector units of the VP2400/10, taking as reference the execution time of the automatically vectorized version. Its value is obtained as follows:

$$\%Vect.Exec. = (TVU/TCPU) * 100 \quad (13)$$

Finally, the column *%Vectorization* provides an idea of the characteristics of the code so as to be executed on a vector machine. Its value is computed according

to the formula

$$\%Vectorization = (T_Orig_Vect/T_Orig) * 100 \quad (14)$$

where T_Orig_Vect is the execution time in scalar mode corresponding to the vectorizable part of the original source code, that is to say,

$$T_Orig_Vect = T_Orig - (TCPU - TVU) \quad (15)$$

The group of columns labeled as T_MV shows the times corresponding to the execution of the optimized version of the code that we have implemented.

Note that we have been able to increase the $\%Vectorization$ from a 50% up to a 94%, approximately. This redounds to a great decreasing of the computational cost of the algorithm. The execution times of the vectorized version for *mesh4* and *mesh5* are presented in Table 7.

Table 7. Execution times for *mesh4* and *mesh5* on the VP2400/10.

	T_MV					
	<i>mesh4</i>			<i>mesh5</i>		
	TVU	TCPU	% Vect.Exec.	TVU	TCPU	% Vect.Exec.
b	08:44:36	11:11:55	72.82%			
l	17:14:28	22:08:59	77.28%	60:38:56	77:36:37	77.93%
g	20:25:24	26:17:54	76.94%			
j	33:45:50	43:38:38	76.75%	68:54:51	88:37:47	77.28%

The aim of the work was not only to reduce computing time but also to test in practice the convergence of the finite element space discretization. So, the finer grids *mesh4* (38400 triangular finite elements) and *mesh5* (76800 triangular finite elements) were considered. For both new meshes, the analysis of huge partial computing times prevented us from executing the scalar code. In Figs. 4 and 5 the pressure and gap approximation profiles are presented for *mesh4* and *mesh5*, respectively, in an appropriate scale. In both figures, the relevant real parameter is the imposed load which is taken to be $\omega = 3$.

Graphics included in Figs. 4 and 5 represent the low boundary of the domain of the problem (see Fig. 1) on the x axis. There are two sets of curves. The mountain-like one represents the pressure of the lubricant in the contact zone. The parabola-like one represents the gap between the two surfaces in contact.

Next, in order to verify the expected qualitative behaviour already observed in previous tests with *mesh3* when increasing the charge, we considered *mesh4* for the datum $\omega = 5$. In Fig. 6 pressure and gap profiles were computed for $\omega = 5$.

6 Conclusions

From the numerical viewpoint, the practical convergence validation of a new algorithm has been performed with the help of the vectorization techniques applied

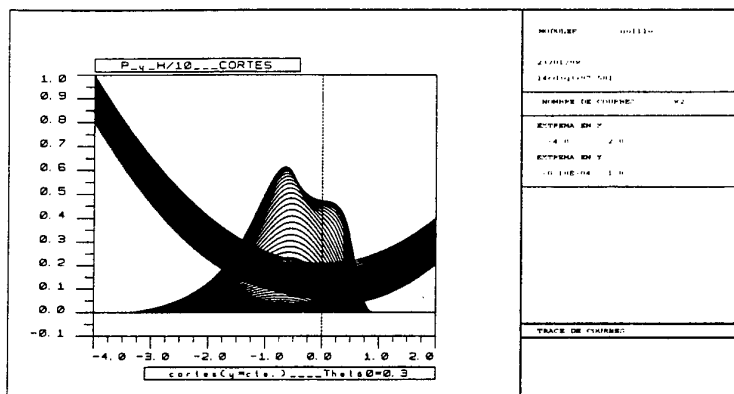


Fig. 4. Pressure and gap profiles for $\omega = 3$ and $\theta_0 = 0.3$ with *mesh4*.

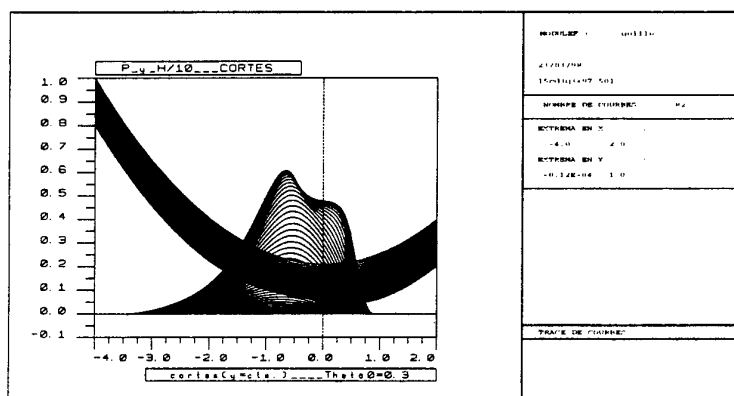


Fig. 5. Pressure and gap profiles for $\omega = 3$ and $\theta_0 = 0.3$ with *mesh5*.

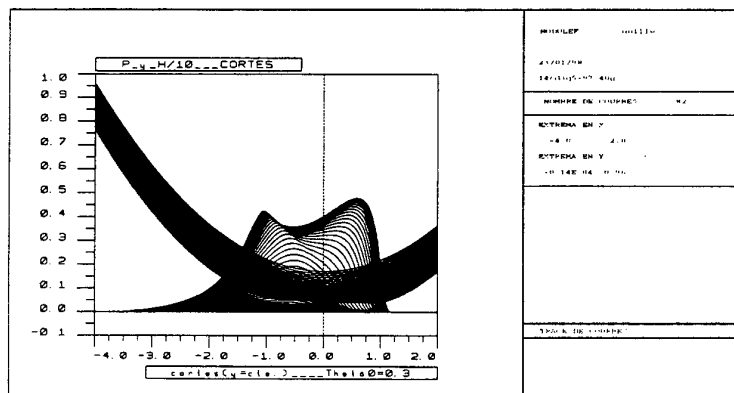


Fig. 6. Pressure and gap profiles for $\omega = 5$ and $\theta_0 = 0.3$ with *mesh4*.

to the ancient scalar code. Moreover, the accuracy of the approximation has been increased with the introduction of finer grids (*mesh4* and *mesh5*) that confirm the expected results for the numerical method implemented by the algorithm.

We have developed a vector version where at about the 94% of the original source code has been vectorized. As compensation, our vector version has partially lost the generality of the original programme and has been implemented in a manner that is dependent on the architecture of the machine.

On the other hand, a first approach to a parallel version of the algorithm is being developed. As an example, we have presented a parallel implementation of one functional block of the algorithm, being noticeable the *superspeedup* obtained in some cases.

Nowadays, we are refining the parallel version so as to reduce the execution time corresponding to the multipliers loop and so as to execute a final version on a multiprocessor architecture like the Cray T3E.

References

1. Arenaz M.: *Improving the performance of the finite element software for an elastohydrodynamic punctual contact problem under imposed load*. (in spanish) Master Thesis, Departament of Electronics and Systems, University of A Coruña (1998).
2. Bayada G., Chambat M.: *Sur Quelques Modelisations de la Zone de Cavitation en Lubrification Hydrodynamique*. J. of Theor. and Appl. Mech., **5**(5) (1986) 703-729.
3. Bermúdez A., Moreno C.: *Duality methods for solving variational inequalities*. Comp. Math. with Appl., **7** (1981) 43-58.
4. Cameron A.: *Basic Lubrication Theory*. John Wiley and Sons, Chichester (1981).
5. Durany J., García G., Vázquez C.: *Numerical Computation of Free Boundary Problems in Elastohydrodynamic Lubrication*. Appl. Math. Modelling, **20** (1996) 104-113.
6. Durany J., García G., Vázquez C.: *A Mixed Dirichlet-Neumann Problem for a Non-linear Reynolds Equation in Elastohydrodynamic Piezoviscous Lubrication*. Proceedings of the Edinburgh Mathematical Society, **39** (1996) 151-162.
7. Fujitsu España: *FVPLEB user's guide*. (1992).
8. Fujitsu España: *FORTTRAN77 EX/VP user's guide*. (1992).
9. García G.: *Mathematical study of cavitation phenomena in elastohydrodynamic piezoviscous lubrication*. Ph.D. Thesis, University of Santiago (1996).
10. Lubrecht A.A.: *The Numerical Solution of the Elastohydrodynamic Lubricated Line and Point Contact Problems using Multigrid Techniques*. Ph.D. Thesis, Twente University (1987).
11. Wolfe M.: *High Performance Compilers for Parallel Computing*. Addison-Wesley Publishing Company (1996).

Distributed simulation strategies of graphite electrode forming process

Danielewski M, Bożek B¹, Holly K², Myśliwiec G, Sipowicz J, Schaefer R³

¹University of Mining and Metallurgy, Mickiewicza 30, 30-059 Kraków

²Institute of Mathematics, Jagiellonian University, Reymonta 4, Kraków, POLAND

³Institute of Computer Science, Jagiellonian University, Nawojki 11, 30-072 Kraków, POLAND

Abstract. The simulation of complex mass transport process, namely the diffusion affected flow of high viscosity fluid in transient press has been performed. Massive parallel algorithms were utilized at crucial phases of simulation (matrix formulation, linear solver). The advanced, double feedback task management system have increased efficiency of the distributed implementation of algorithms mentioned above. The first feedback works outside of the application and couples the network diagnostic system with task allocation and migration mechanisms. The second one deeply affects the numerical algorithm, adopting dynamically the mesh partitioning in each iteration. The stochastic performance forecast is used in task allocation policies.

Motivation

Today's dynamic simulations of transport processes are powerful and widely used, e.g. by the space industry and in the advanced control systems on modern plants. Yet, we still make little use of dynamic modeling of complex thermochemical processes. Apparently, there is a growing demand for the more advanced modeling of the real, practical systems. Contrary to an extensive research on modeling of injection molding [1], the modeling of three-dimensional flow fields in transient presses is relatively little known [2]. Such units are widely used, e.g., in ceramic as well as carbon industry, to form various elements that later undergo the thermal and/or ageing treatments. The medium is usually multi component and multi phase homogeneous slurry, that shows very complex properties. Modeling of the flow in such process represents several major challenges since the flow is inherently transient, includes a free surface and fluid is moving through irregular extrusion die.

The above numerical simulation requires huge CPU and RAM resources, so only massively parallel algorithms can be utilized. Because of limited access to fast multiprocessor installations, we decided to develop a distributed implementation, which is dedicated to local network of workstations and medium-size LAN servers.

Methods that can dynamically adapt a distributed application to a current state of network environment significantly decrease its execution time.

Mathematical statement and physical description

This work will show the simulation of complex mass transport process namely, the diffusion affected flow of high viscosity fluid in the transient press. The mathematical model of the process allows to examine the effects of the nonuniform heating of the press wall (temperature dependent viscosity), the geometry of an extrusion die and the influence of lubricant of the fluid wall friction. An obvious simplification is an assumption that slurry is Newtonian compressible fluid.

We will study the behavior of fluid contained in time dependent domain $\Omega(t) \subset \mathbb{R}^3$ during the period $t \in [0, t_{\text{limit}}]$. Its boundary $\partial\Omega(t)$ is a disjoint sum $\partial\Omega(t) = \partial_{\text{rig}}^k \Omega(t) \cup \partial_{\text{free}}^k \Omega(t) \cup \partial_{\text{free}}^u \Omega(t)$ of three regular surfaces: known rigid boundary, known free boundary and unknown free boundary respectively.

We are looking for the following unknowns: $v: \bigcup_t \{t\} \times \Omega(t) \rightarrow \mathbb{R}^3$ velocity of the fluid and $p, \rho, T, g: \bigcup_t \{t\} \times \Omega(t) \rightarrow \mathbb{R}$ which represent fluid pressure, density, temperature and lubricant density respectively. They should satisfy three basic conservation laws:

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \text{div}(\rho v) &= 0 && \text{mass conservation law,} \\ \rho \left(\frac{\partial v}{\partial t} + \partial^{(v)} v \right) &= \text{Div} \mathfrak{S} + \rho b, && \text{momentum conservation law} \\ \mathfrak{S} &= \left\{ \frac{v}{2} \left((v_{i,j} + v_{j,i}) - \frac{2}{3} (\text{div} v) \delta_{ij} \right) - p \delta_{ij} \right\} && \text{(the Navier-Stokes system),} \\ \frac{\partial E}{\partial t} + \text{div}(Ev) &= \text{div}(\mathfrak{S}v + \Theta \text{grad} T) && \text{law of total energy conservation.} \\ \frac{\partial g}{\partial t} + \text{div}(gv) - D \text{grad} g &= 0 && \text{conservation law of the} \\ &&& \text{lubricant mass} \end{aligned}$$

where $v = v(T(t, x))$, $\Theta = \Theta(T(t, x))$ and $D = D(T(t, x))$ are viscosity and thermal diffusivity of the fluid and diffusion coefficient of the lubricant, $b = b(t, x)$ denotes external mass forces (e.g. gravity) assumed to be well known functions of time $t \in [0, t_{\text{limit}}]$ and position $x \in \Omega(t)$. Moreover:

$$\begin{aligned} E &:= \frac{3}{2} \left(p + \frac{1}{2} \rho |v|^2 \right) && \text{is the energy density of the fluid,} \\ p &= p(\rho, T) = C_1 (\rho - \rho^*)^\gamma + C_2 \rho T, && \text{will be utilized as the equation} \\ \gamma > 1, C_1, C_2, \rho^* &> 0 && \text{of state.} \end{aligned}$$

We assume the following initial conditions: $v(0) = v_0$, $p(0) = p_0$, $\rho(0) = \rho_0$, $T(0) = T_0$, $g(0) = g_0$ on $\Omega(0)$; and for each $t \in [0, t_{\text{limit}}]$ the following boundary conditions:

- i) $T(t) = T_0(t)$ on $\partial\Omega(t)$,
- ii) $\frac{\partial g}{\partial \mathbf{n}_t}(t) = 0$ on $\partial_{\text{rig}}^k \Omega(t)$,
- iii) $(v(t)|\mathbf{n}_t)$ vanish on $\partial_{\text{rig}}^k \Omega(t)$ in coordinate system which is stiffly attached to the press wall,
- iv) $\mathfrak{I}(t)\mathbf{n}_t = -p_{\text{ext}}(t)\mathbf{n}_t$ on $\partial_{\text{free}}^u \Omega(t)$,
- v) $\mathfrak{I}(t)\mathbf{n}_t = -(\mathfrak{I}(t)\mathbf{n}_t|\mathbf{n}_t)\mathbf{n}_t$ is perpendicular to $v(t) - (v(t)|\mathbf{n}_t)\mathbf{n}_t$ on $\partial_{\text{rig}}^k \Omega(t)$.

where \mathbf{n}_t denotes the unit outward normal to the surface $\partial\Omega(t)$. Detailed formulation of the above problem can be found elsewhere in [3].

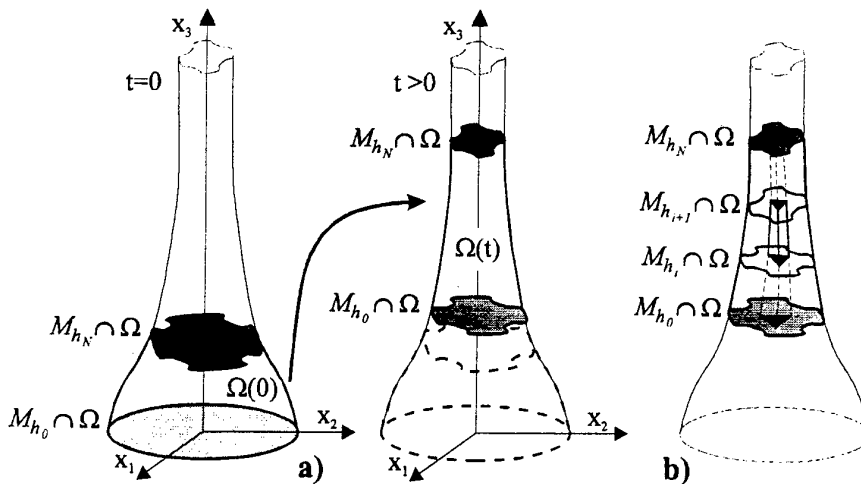


Fig. 1 Schematic view of: a) the evolution of fluid mass in the transient press, b) the basic cell of triangulation in $\Omega(0)$.

Discrete FE/FD approach

We consider an arbitrary domain $\Omega \subset \mathbb{R}^3$ with regular boundary such that $\mathbb{R}e_3 \subset \Omega$ and one-parameter family $\{M_h\}_{h \in \mathbb{R}}$ of planes which are orthogonal to the axis $\mathbb{R}e_3$ such that $M_h := \{x \in \mathbb{R}^3 : (x - he_3|e_3) = 0\}$. Moreover we assume that

$$\forall h \in \mathbb{R}, \forall \xi \in \Omega \quad T_\xi M_h + T_\xi \partial\Omega = \mathbb{R}^3$$

where $T_\xi \partial \Omega$ is a plane tangent to $\partial \Omega$ in the point ξ . We assume that we know two functions $h_{\min}, h_{\max}: [0, t_{\text{limit}}] \rightarrow \mathbb{R}$ such that $h_{\min}(t) < h_{\max}(t)$, for each $t \in [0, t_{\text{limit}}]$ (see Fig. 1.a). Let us define

$$\Omega(t) := \Omega \cap \{x \in \mathbb{R}^3, x = (x_1, x_2, x_3) : h_{\min}(t) \leq x_3 \leq h_{\max}(t)\}, \quad t \in [0, t_{\text{limit}}] \quad (1)$$

and distinguish the sequence of points $h_{\min}(0) =: h_0 < h_1, \dots, < h_{N-1} < h_N := h_{\max}(0)$. Using the algorithm of two dimensional (flat) Delunay triangulation [4,5] we attempt to triangulate the first layer $M_{h_0} \cap \Omega$. Next we copy this triangulation to remaining layers $M_{h_i} \cap \Omega, i = 1, \dots, N$ solving the Dirichlet problem for two-dimensional Laplace equation. Then, using nodal points from two consecutive layers we build basic cells of triangulation (see Fig. 1.b). Each of these cell is spread to six simplexes of cubic triangulation in a canonical way. The initial mesh is transformed to the mesh covering $\Omega(t)$ for an arbitrary time instance $t > 0$. Mesh nodes take new positions, but triangulation topology keeps unchanged. Let us denote by \wp the set of node labels, associated with the initial mesh. Thanks to the constant topology, labels remain valid at $t > 0$.

To solve the presented differential problem we use the Faedo-Galerkin method with respect to the spatial variables (see e.g. Thomee [6] and Lions, Magenes [7]). We will use the family $\{X_t\}, t \in \mathbb{R}_+$ of finite dimensional spaces spanned by Lagrange 1st degree splines $\varphi_P: \bigcup_t \{t\} \times \Omega(t) \rightarrow \mathbb{R}$ which are affine on every simplex and takes 1 at $x_P \in \Omega(t), P \in \wp$ (the current position of P^{th} node) and 0 at $x_Q \in \Omega(t), Q \in \wp, P \neq Q$ for each $t \in [0, t_{\text{limit}}]$. We look for approximate solution in a form:

$$\sum_{P \in \wp} \lambda_P(t) \varphi_P(t, x) \in X_t, \quad t \in [0, t_{\text{limit}}] \quad (2)$$

where $\lambda_P(t) \in \mathbb{R}$ represent approximate nodal values of $\{v_i(t)\}, i = 1, 2, 3, p(t), \rho(t), T(t), g(t)$. After spatial integration we may pass to the initial problem for the system of ODE's:

$$L(t) \dot{\Lambda}(t) = F(t, \Lambda(t)), \quad \Lambda(0) = \Lambda_0, \quad \Lambda(t) := \{\lambda_P(t)\}_{P \in \wp} \quad (3)$$

Next we can solve the above system using one of the multi-step methods such as Adams-Bashforth [8] method of third degree:

$$L(t_i)(\Lambda_{i+1} - \Lambda_i) = \frac{h}{12} (23F(t_i, \Lambda_i) - 16F(t_{i-1}, \Lambda_{i-1}) + 5F(t_{i-2}, \Lambda_{i-2})) \\ \Lambda_i := \Lambda(t_i), \quad i = 0, 1, 2, \dots, \quad 0 < t_i < t_j \quad \text{for } i < j \quad (4)$$

Finally we consecutively solve the linear algebraic system

$$L_i \Lambda_{i+1} = R_i, \quad i = 1, 2, \dots, \quad (5)$$

where

$$L_i := L(t_i), \quad R_i = \frac{h}{12} (23F(t_i, \Lambda_i) - 16F(t_{i-1}, \Lambda_{i-1}) + 5F(t_{i-2}, \Lambda_{i-2})) + L(t) \Lambda_i.$$

The overall scheme of distributed computation

Due to the large size of the spatial mesh and the nonstationary character of the solution $\Lambda(t)$ as well as the solution's domain $\Omega(t)$, the problem needs effective and flexible solving methods. First task which should be performed after input data are entered is the initial mesh generation in the initial domain $\Omega(0)$ (see Fig. 2, Stage I). Thanks to the special mesh structure, which traces the $\Omega(t)$ deformation, the initial mesh topology is valid during the whole computation time. Only new node positions should be determined at each t . They can be computed sequentially, due to the low computational complexity.

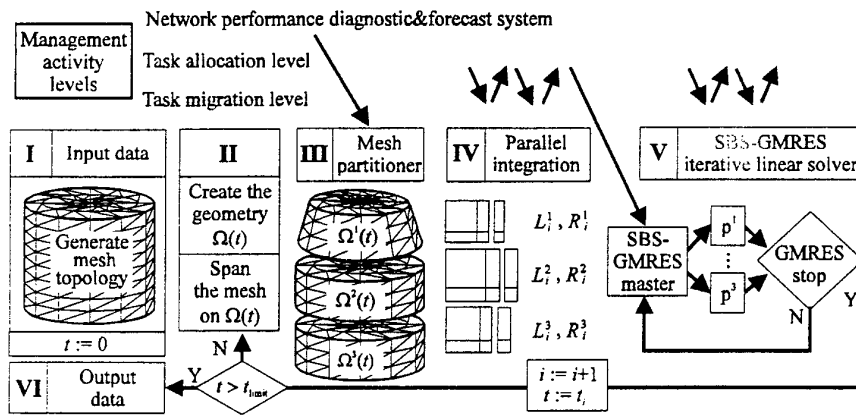


Fig. 1 The general management scheme

Main operations which have to be performed at each time steps $t_i, i = 1, 2, \dots$ are L_i matrix calculation and calculation of the right-hand side vector R_i , as well as the solution of linear system (5) which represents the finite difference scheme to be used. The first task has the square and the second one the cube serial computational complexity with respect to the number of degrees of freedom (d.o.f.).

The course of decreasing the computational time is to parallelize the above tasks at each time step $t_i, i = 1, 2, \dots$. It can be obtained if the mesh spanned on $\Omega(t_i)$ can be partitioned into submeshes covering connective, disjoint subdomains $\Omega^s(t_i), s=1, \dots, S$. Another usual need is to keep the number of degrees of freedom laying on subdomain interfaces $\Omega^r(t_i) \cap \Omega^s(t_i), r \neq s$ on sufficient low level. The regular, constant mesh topology allow us to split $\Omega(t_i)$ into subdomains composed of the arbitrary number of element „layers”, perpendicular to the press axis. The main advantage of this method is the linear growth of interface d.o.f. only with respect to subdomain number and do not depend on subdomain sizes.

Matrices L_i and R_i can be computed in parallel in each subdomain, according to SBS strategy, and then system (5) can be transformed to the Schur Complement form (cf. Papadrakakis [9]). Finally, the Schur Complement can be solved using parallel GMRES algorithm with distributed RAM utilization (see Golub [10]).

The described above parallel computation is implemented on the distributed MIMD architecture \mathcal{H} (e.g. network of workstations) as the set of intercommunicating processes.

The complex computation structure forces us to the multilevel managing, which leads to maximizing speedup and economic computer resources utilization (CPU, RAM,...) with respect to dynamic load changes of computer nodes. Managing activities can be applied on three distinct levels:

1. *Network diagnostic & forecast level* at which we decide about the number and size of subdomains $\Omega^s(t_i)$ which will be valid during at least one computational step t_i .
2. *On-line task allocation level* where a task set can be assigned to processor unit (workstation) using various policies, basing on a current load measurement. Above policies will be utilized to initial allocation of matrix computation tasks (see stage IV in Fig. 2) and GMRES tasks (stage V in Fig. 2) at each GMRES iteration step.
3. *Task migration level*, where we can rearrange initial task allocation performed in on-line task allocation level, before application comes to the nearest synchronization point.

Dynamic task allocation strategies

Task migration and simple allocation policies

Dynamic task allocation is performed after each synchronization event, if forthcoming tasks can be processed in parallel. Three strategies can be proposed:

- task are allocated in a cyclic (round-robin) manner. Each machine gets approximately the same number of tasks.
- Serial computational complexity of the group of tasks which are sent to particular machine is proportional to its static benchmarked (nominal) speed. Such solution is suitable for networks which are only slightly loaded and without a significant load disproportion between different machines.
- Serial computational complexity of the group of tasks which is assigned to particular machine is proportional to its *power coefficient* (see next chapter for more precise description) which is currently measured, or determined on the base of Markov load model described later.

Task migration can change task location before the nearest synchronization event, even during the task execution. Only simple migration criterion was utilized: when machine $X \in \mathcal{H}$ has finished its work (become to be idle), and on machine $Y \in \mathcal{H}$

there are at least two processes of our application, then one of them is migrated to X. This policy is especially useful for on-line correction of coarse-grained initial task allocation on machines with the rapid and unpredictable load variability, so it was mainly used in step V (see Fig. 2).

Stochastic dynamics of a computer network, and Markov allocation policy

Key decision of coarse-grained GMRES task forming can be supported by stochastic forecasts based on Markov chain dynamic model of MIMD architecture \mathcal{H} composed of M processor units connected by the asynchronous communication medium (e.g. package-oriented network technology). We distinguish the whole range of physical performance parameter values of each machine (CPU utilization, RAM occupation, load, etc.) on K adjacent subranges, and choose the sequence of time instances $n=1,2,\dots$ (e.g. end of each hour), so we may say, that the particular machine is in the state $j \in \{1,\dots,K\}$ if its performance parameter is in the j^{th} subrange. We assume periodic behavior (e.g. with the 24 hour period) of each performance parameter, so it suffices to identify the finite set of transition matrices $\{P^m(n)\}$, $m=1,\dots,M$ of dimension $K \times K$, such that dynamics can be described by Markov evolution formula:

$$\Pi^m(n) = P^m(n-1)\Pi^m(n-1), \quad m=1,\dots,M \quad (6)$$

where $\Pi^m(n)$ is the probability distribution vector (of dimension K) of the physical parameter in the n^{th} time step for the machine number m . If we know the initial distribution $\Pi^m(\mu)$ we are able to foresee $\Pi^m(n)$ for $n > \mu$ using (6) consecutively. $\Pi^m(\mu)$ may be set as $\{\delta_{j\mu}\}$ if we have observed, that m^{th} machine is in j^{th} state at the initial time step μ . We assume, that each task to be processed in parallel has a computational complexity being a multiple of the *pattern task* complexity. It is true in case of discrete FEM analysis, when single element operations quantify the pattern task size.

In order to pass to more convenient description we introduce new Markov chain $\bar{T}^m(n)$ which describes the mean execution time of pattern task in n^{th} time period on m^{th} machine. This chain bases also on the partitioning on K adjacent subranges, similar as in case of physical performance parameters, such that $\bar{T}^m(n)$ can take K discrete states with this same probability distribution $\Pi^m(n)$. We refer to the paper Onderka, Schaefer [11] for more details of the above model.

Using identified chains $\{\bar{T}^m(n)\}$, $m=1,\dots,M$ the useful *power coefficients* for n^{th} time instance, $n > \mu$ can be determined as:

$$\begin{aligned} \phi_m(\mu, n), \quad \Phi_m(\mu, n) &= \frac{\lambda_m(\mu, n)}{\Lambda(\mu, n)}, \quad \lambda_m(\mu, n) = \left(E_{\Pi^m(\mu)}(\bar{T}^m(n)) \right)^{-1}, \\ \Lambda(\mu, n) &= \sum_{m=1}^M \lambda_m(\mu, n) \end{aligned} \quad (7)$$

where $E_{\Pi^m(\mu)}$ is the expected value operator assuming the initial probability distribution $\Pi^m(\mu)$. We will use the open-loop Markov policy which consists in:

- evaluating $\Pi^m(\mu)$ at the beginning of each time loop $t_i, i = 1, 2, \dots$ (see stage III on Fig. 2) and computing power coefficients $\phi_m(\mu, \mu + 1)$ for $m = 1, \dots, M$.
- decomposing $\Omega(t)$ into disjoint subdomains $\{\Omega^s(t)\}, s = 1, \dots, S$ for $S = rM$ (computational mesh „layers”) for some $r \geq 1$, such that the overall computational complexity of tasks (matrices computation tasks and GMRES tasks) associated with subdomain numbers $(m-1)r, (m-1)r+1, \dots, mr$ is proportional to the power coefficient $\phi_m(\mu, \mu + 1)$ of m^{th} machine.
- allocating tasks number $(m-1)r, (m-1)r+1, \dots, mr$ to m^{th} processor unit.

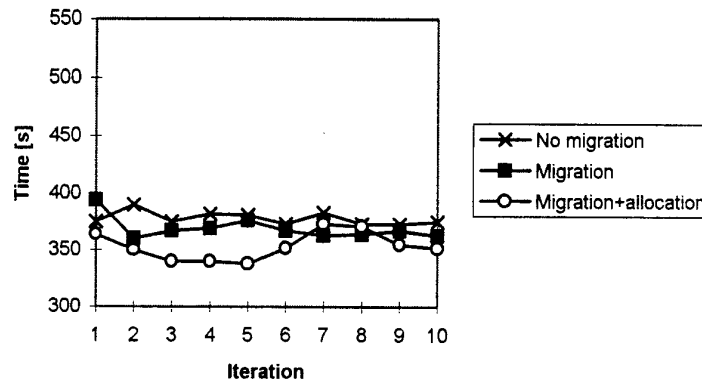


Fig. 1. Test result, case no. 1

Test results

Various combinations of task management methods have been implemented and tested for simple press geometry, without local adoption and computational mesh adoption. A possibility of flexible mesh partitioning and hardware environment enabling tasks migration implies no need for advanced forecast methods (since improper allocation can be easily improved). However the simulation system will be applied further for press of more complicated geometry and h-p mesh adoption, which cause the significant growth of computation and memory complexity and worse granularity of elementary tasks. Sophisticated stochastic control justified in this case, as in example presented in [12].

Below we present results of three different simple task allocation strategies used during tests:

1. all machines get the same number of tasks (*No migration* on the charts),
2. all machines get initially the same number of tasks but their location can change during the execution using the migration criterion described above (*Migration*),
3. similar to point 2, but the initial task distribution is determined basing on the result from the previous time step (*Migration+allocation*).

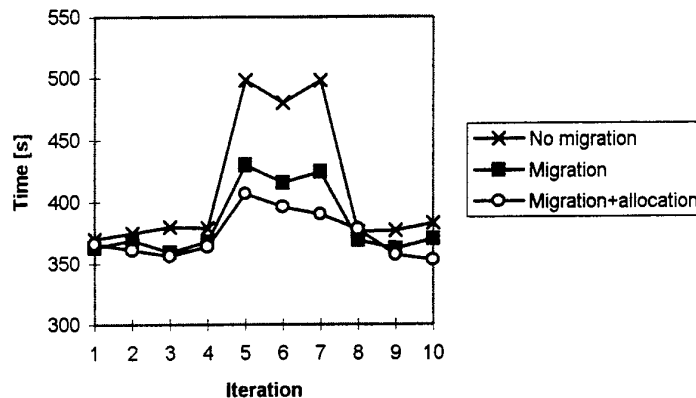


Fig. 1. Test result, case no. 2

The problem containing 48861 d.o.f. has been solved. Results for first 10 time steps for two test cases are presented below in the Fig. 3 and 4. In the first case all machines were not loaded (except, of course, for some system processes/daemons). In the second case, machine #3 was additionally slightly loaded by two processes (consuming about 30% of CPU) during iterations 5-7. In the first case, all results were similar, with a little advantage of the migration-based strategies. In the case no. 2, migration policies (especially *Migration+allocation*) gave much better results than the simple round-robin strategy.

Table 1. Test case #2, strategy *No migration*

iteration	time [s]	number of tasks on machine			
		#1	#2	#3	#4
1	370	38/-0/+0	37/-0/+0	37/-0/+0	38/-0/+0
2	375	38/-0/+0	37/-0/+0	37/-0/+0	38/-0/+0
3	380	38/-0/+0	37/-0/+0	37/-0/+0	38/-0/+0
4	379	38/-0/+0	37/-0/+0	37/-0/+0	38/-0/+0
5	498	38/-0/+0	37/-0/+0	37/-0/+0	38/-0/+0
6	480	38/-0/+0	37/-0/+0	37/-0/+0	38/-0/+0
7	498	38/-0/+0	37/-0/+0	37/-0/+0	38/-0/+0
8	376	38/-0/+0	37/-0/+0	37/-0/+0	38/-0/+0
9	377	38/-0/+0	37/-0/+0	37/-0/+0	38/-0/+0
10	383	38/-0/+0	37/-0/+0	37/-0/+0	38/-0/+0

More detailed results for the second test case are presented in Tables 1-3. For each machine are shown: initial number of tasks / number of tasks migrated from the machine / number of tasks migrated to the machine.

Table 1. Test case #2, strategy *Migration*

iteration	time [s]	number of tasks on machine			
		#1	#2	#3	#4
1	363	38/-0/+4	37/-0/+12	37/-13/+0	38/-8/+5
2	369	38/-0/+4	37/-0/+12	37/-12/+0	38/-8/+4
3	359	38/-0/+4	37/-0/+5	37/-9/+0	38/-2/+2
4	368	38/-0/+4	37/-0/+12	37/-12/+0	38/-8/+4
5	430	38/-0/+5	37/-0/+10	37/-0/+3	38/-18/+0
6	416	38/-0/+6	37/-0/+8	37/-0/+3	38/-17/+0
7	425	38/-0/+6	37/-0/+9	37/-0/+3	38/-18/+0
8	369	38/-0/+5	37/-0/+11	37/-13/+0	38/-8/+5
9	362	38/-1/+4	37/-0/+12	37/-12/+0	38/-7/+4
10	370	38/-3/+3	37/-0/+11	37/-10/+0	38/-7/+6

Table 2. Test case #2, strategy *Migration+allocation*

iteration	time [s]	number of tasks on machine			
		#1	#2	#3	#4
1	366	38/-0/+5	37/-0/+10	37/-11/+0	38/-8/+4
2	361	43/-14/+0	47/-0/+7	26/-0/+8	34/-6/+5
3	356	29/-0/+12	54/-5/+4	34/-11/+0	33/-5/+5
4	364	41/-8/+0	53/-2/+0	23/-0/+10	33/-4/+4
5	407	33/-0/+7	51/-0/+5	33/-0/+2	33/-14/+0
6	396	40/-5/+4	56/-0/+5	35/-12/+0	19/-0/+8
7	390	39/-0/+3	61/-4/+0	23/-0/+7	27/-8/+2
8	378	42/-5/+4	57/-6/+1	30/-4/+0	21/-0/+10
9	357	41/-8/+0	52/-0/+2	26/-0/+1	31/-0/+5
10	353	33/-0/+6	54/-3/+0	27/-0/+0	36/-4/+1

Scalability of the whole computational scheme

Scalability of the parallel, distributed application with respect to the MIMD architecture \mathcal{H} can be understood as the relationship $S \subseteq \mathcal{R} \times \mathcal{N}$, where $\mathcal{R} \subseteq \mathcal{H}^i$ is the set of \mathcal{H} resources (CPUs, RAM, etc.), $\mathcal{N} \subseteq \mathcal{H}^l$ is the set of application requirements (computation time, etc.), k, l are natural numbers. We may study also *conditional*

scalability as the partial relationship $S_c \subset S$ with some fixed parameters from both \mathcal{R} and \mathcal{N} .

We may consider in case of GMRES-SBS application $k=l=2$, $\mathcal{R}=\{(N, R)\}$, $\mathcal{N}=\{(T, n)\}$, where N , R , n , T are the number of processors, RAM per node, number of degrees of freedom and the time of parallel computations respectively. Assuming again that the number of GMRES iterations is proportional to the \sqrt{n} (see [13]), we have:

- N is proportional to $n^{\frac{3}{2}}$, by $T=\text{const.}$,
- N is proportional to $n-2\sqrt{n}+1$ assuming the constant memory per node,
- $N^2 + AN = (\sqrt{n}-1)(B-C(\sqrt{n}-1))$ by the constant efficiency, where A , B , C

are positive constants, which depend on the transmission speed, processor arithmetic and the amount of computation for the single element in the single GMRES iteration and for the local stiffness matrix formulation and inversion.

See [14] for the proof of the above proposition.

Mesh partitioning

Proper partitioning of a mesh has a great influence on the resulting computation performance. Every slave task has a computational complexity of $O(n_s^2)$ (where n_s is the number of internal d.o.f. in $\Omega^s(t_i)$) and the size of the global interface is (only) a linear function of subdomain number. Therefore, at the first sight, the more subdomains we have, the shorter is computation time. In practice, however, too large number of tasks requires a lot of network transmissions which can „consume” all profits gained from the lower theoretical complexity of problem.

Our tests show that there exist few points (possible single) of „equilibrium” between increase of computation time caused by too big size of subdomains and performance degradation introduced by too frequent and intensive network communication. At that points the total execution time (as a function of tasks number) has a minimum. Moreover, it seems that they occupy narrow range so in practice they can be easily found experimentally. Naturally, they can be different for different computational environments or different conditions (e.g., load, network traffic). Nevertheless, we think it is enough to determine the „optimal” number of tasks once, at the beginning of computations.

Below you can see execution times of one SBS iteration for a mesh composed of 24461 d.o.f. Tasks has been allocated equally to all available machines; no migration was applied.

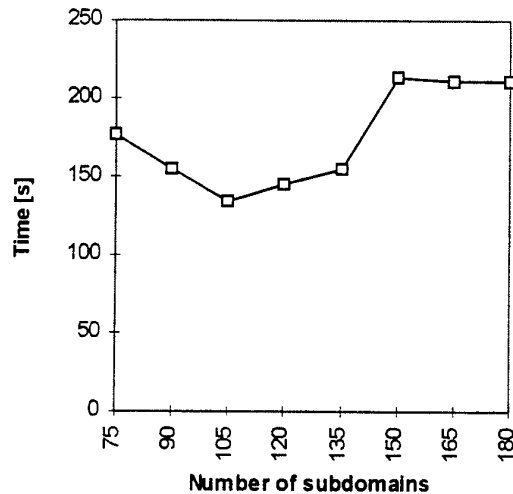


Fig. 1. Execution time for various mesh partitionings

Since, as the tests show, in usual case a mesh should be divided on at least several tens of domains which are distributed among only several machines, we always partition a mesh into subdomains of equal (almost equal, in fact) sizes. Such approach simplifies task management (e.g., it is easier to predict execution times).

Computational environment

All tests presented in this paper has been run using a network composed of:

- 4 Sun SparcStation (4, 5 and 20 models) machines with various RAM amount (24-128 MB) running SunOS 4.1.4 — for slave processes,
- 1 PC (Pentium 66) running Linux — for master and control processes.

MPVM 3.3.4 [15] as a communication and migration tool has been used.

Target version of system described in this paper is developed according to Object Oriented (OO) paradigm, using CORBA-based library (Chorus [16]). This version have been run in heterogeneous environment composed of Sun SparcStations ELC, Sun 490, SGI Origin 200 and exhibits a quite similar effectiveness as PVM version. CORBA based distributed environments allow object mobility which is equivalent to task migration in MPVM, offering much more portability with respect to hardware platform and operating systems. This possibility will be soon included in our target version of the system.

Conclusions

- i. The simulation of the forming process will give the ultimate answer for the best manufacturing parameter combinations to reduce the production cost.
- ii. The increased speed of computations will allow to implement process simulation as the guideline for mixing, recipe parameters' adaptation and optimization of the formation conditions in a production plant.
- iii. Task migration can dramatically improve control effect in rapid changes of computer performance during one time iteration step (see case #2, fig 4)
- iv. The application of stochastic policy is meaningless in case of fine granularity of elementary tasks, because migration can easily improve improper initial allocation. However stochastic policy is best suited to the coarse-grained problems with huge computational and memory complexity and periodically loaded network. It appears mainly in complex geometry CAE problems solved by h-p adaptive finite element method (see e.g. [12, 14].
- v. Final number of tasks (after migration) which is run on the machine during particular iteration can be used to determine the power coefficient for the next step (see case #2, *migration + allocation*)
- vi. Partial scalability tests (number of subdomains vs. CPU, RAM and network) prove that there exists optimal partitioning rate for the current CAE parallel problem and distributed computer environment (see. Fig. 5).

References

1. Manzione L. T. ed.; Application of Computer Aided Engineering in Injection Molding, Hanser Publ., 1987.
2. Born M et al.; Carbon, 30, 141, 1992.
3. Danielewski M, Holly K, Bozek B, Bednarsz S, Golec S and Filipek R; Dynamics of the graphite electrode forming process, Univ. of Mining and Metallurgy, Cracow, 1998, Rep. 1246/98.
4. Bozek B, Holly K, Jaskólski J; Variance methods for thermo load of elements of IC engine, 20th International On Combustion Engines (CIMAC 1993), London 1993, D74.
5. Holly K, Mosurski R; An automatic triangulation of an arbitrary flat domain, Opuscula Mathematica, Vol. 17, 1997, pp. 23 - 32.
6. Thomée V; Galerkin Finite Element Methods for Parabolic Problems, Springer Verlag 1984.
7. Lions J.-L., Magenes E; Problèmes aux limites non homogènes et applications, vol. 1 et 2, Paris, Dunod, 1968.
8. Burden R. L, Faires J. D; Numerical Analysis third edition, Boston, Prindle, Weber & Schmidt, 1985.

9. Papadrakakis M, Bitzarakis S; Domain decomposition PCG-methods for serial and parallel processing, *Computing Systems in Engineering*, 1995.
10. Geist A. and others; PVM 3 User's Guide and Reference Manual, Oak Ridge National Laboratory, 1994.
11. Onderka Z, Schaefer R; Markov chain based management of large scale distributed computations of earthen dam leakages, *Lecture Notes in Computer Science* 1215, Springer Verlag 1996, pp. 49-64.
12. Myśliwiec G, Sipowicz J, Schaefer R; Control activities in Message Passing Environment, *Lecture Notes in Computer Science* 1332, Springer Verlag 1997, pp. 143-150.
13. Barragy E, Carey G.F, Van de Geijst R; Performance and Scalability of Finite Element Analysis for Distributed Parallel Computation, *Journal of Distributed and Parallel Computing* 21, 1994, pp. 202-212.
14. Flasiński M, Schaefer R, Toporkiewicz W; Scalability in Concurrent Discrete Analysis of Structures. *Proc. of PCCMM'97*, Vol. 1, pp. 379-386, Poznań 1997.
15. Casas J., Clark D. L., Konuru R., Otto S.W., Prouth R.M., Walpole J: MPVM: A Migration Transparent Version of PVM, *Computing Systems*, vol. 8, No. 2, pp. 171-216, 1995
16. The Common Object Request Broker Architecture and Specification; Revision 2.0, Object Management Group, 1995.

Experimental Analysis of a Parallel Quicksort-Based Algorithm for Suffix Array Generation

Autran Macêdo¹, Marco Antônio Cristo¹, Elaine Spinola Silva¹, Denilson Moura Barbosa¹, João Paulo Kitajima¹, Berthier Ribeiro¹, Gonzalo Navarro², and Nivio Ziviani¹

¹ Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte, MG - BRAZIL
latin@dcc.ufmg.br

² Departamento de Ciencias de la Computación
Universidad de Chile
Santiago - CHILE
gnavarro@dcc.uchile.cl

Abstract. This paper presents experiments performed with an implementation of a quicksort-based parallel indexing algorithm. Besides the expected reduction in execution time, it was observed that the word frequency distribution of the input textual database has a strong influence on performance. Communication and computational load balances are achieved by processing the same quantity of text on each processor. This effectively occurs due to the auto-similar feature of texts, verified experimentally in this work. Also, as seen by the experiments, the auto-similarity of the word frequency distribution implies that this distribution is independent of the text size. In terms of implementation, the knowledge *a priori* of this word frequency may improve the indexing time by eliminating certain parts of the algorithm.

Keywords: Parallel Processing, Information Retrieval, Index Generation, Auto-Similarity, Message Passing.

1 Introduction

Information retrieval is a research area of growing interest by the scientific community. One of the most relevant research field in that area is the string search in textual databases. This string search involves not only the database query, but also the database indexing and the user interface [1]. In the case of information retrieval in Internet homepages, the search process may involve also the automatic scan of World Wide Web (WWW) sites and the download of these homepages for further indexing.

Index generation time is critical. It has at least the sequential complexity in the order of the database size, since all the words may be indexed. In this

sense, parallel strategies can be devised in order to reduce the index generation time. The algorithm proposed in [2] is based on a quicksort approach, where the textual database is partitioned through processors interconnected by a fast network [3]. The index structure is based on a suffix array [1].

The goal of this paper is to present some experimental results of the first version of the algorithm implementation. It was observed that the word frequency distribution in textual databases plays an important role on the program performance. The following Section describes the parallel index generation algorithm. Next, experiments on index generation are presented. The influence of text characteristics is discussed and followed by some conclusions.

2 The Parallel Generation of Suffix Arrays

Searching a large full text for user specified patterns is a time consuming task which requires special indexing schemas. A *suffix array* (or *PAT array*) [1] is a linear structure composed of pointers to every suffix in the text (since the user normally is allowed to query on words, it is customary to index only word beginnings). These index pointers are sorted according to a *lexicographical ordering* of their respective suffixes and each index pointer can be viewed simply as the offset (counted from the beginning of the text) of its corresponding suffix in the text. To find the user patterns, binary search is performed on the array.

The central idea of the parallel algorithm is as follows, considering a fast network of independent computers [3]. Imagine the final result of the process: the global sorted suffix array. If that array is cut in b similarly-sized portions (which is called *slices*), what the algorithm does is to assign a slice to each processor and make it sort that slice. Originally, each processor contains some elements of each slice.

An α -percentile is the value at position αn in the global sorted suffix array. For example, the $1/r$ -percentile is the element at position b . Our algorithm partitions the data to be worked on by each processor by finding the percentiles $1/r, 2/r, \dots, (r-1)/r$ (r is the number of processors). An alternative definition for slice is: the portion of the global suffix array between two consecutive (i/r) -percentiles.

The algorithm proceeds in four steps:

- **Step 1** - One master processor splits the text into pieces of same size and distributes them among slave processors;
- **Step 2** - Each processor builds internally its local suffix array and determines its local percentiles;
- **Step 3** - The processors cooperate to find the r global percentiles. This defines the part of each slice stored at each processor;
- **Step 4** - The processors engage in a distribution process so that every processor gets the part of its slice stored on any other processor;
- **Step 5** - Each processor completes internally the sorting of its slice.

Consider a text T . T is split into pieces of the same size according with the number of processors involved in the generation of the index, so that each processor has a part of the text (step 1). Each processor generates its local suffix array (step 2). In order to generate the global suffix array, the processors find the percentiles of its local suffix array and broadcast them to the others to determine the global percentiles (step 3). After that, each processor is able to know which part of its suffix array belongs to itself and which parts belongs to their partners. The global all-to-all communication is performed (step 4). Finally, each processor sorts its local suffixes (step 5). The concatenation of local suffix array (of all processors) leads to the global index suffix array.

It can be noticed that the steps 4 and 5 are time dominant, due to suffix arrays broadcast and the need of I/O operations. Figures 1 and 2 show this fact. Figure 1 presents two turning points (phases 1 and 5 at 8 processors) due to contention of disk. It can be argued why not just parallelize only these steps. The answer is the availability of primary memory. The aim of this study is the index generation of very large files (order of GigaBytes). A single computer to perform steps 1, 2 and 3 could not support all text in primary memory, without avoiding page faults.

In this way, the complexity of the algorithm (see [2] for details), in the average case, is

$$O(b \log n)I + O(r^2 \log b + b)C = O(b \log n)I + O(b)C$$

where n is the text size and b is the slice size. I is the computation unit cost and C is the communication cost.

3 Experimental Analysis

The following Sections present the experimental environment and measures concerning execution time and load balancing. Parallel quicksort is typically a scalable strategy: a reduction of the execution time is expected. However, for parallel indexing, the characteristics of the input textual database also influence strongly the program performance.

3.1 Experiments

Two message passing parallel machines are being used in the implementation of the algorithm. One machine is in the CENAPAD - MG/CO, a brasilian supercomputer center, located at UFMG (Federal University at Minas Gerais); the other one is in the LMC (Laboratory of Modeling and Calculus), located at IMAG (Mathematics Institute of Grenoble) in France. The computer of CENAPAD - MG/CO is an IBM SP with 41 nodes and 48 processors at 120 MHz with memory ranging from 256 MegaBytes (MB) to 1 GigaBytes (GB) of primary memory. The network is a switch at 155 MB/s and all processors share a single disk system. The computer in LMC is an IBM SP with 32 processors at

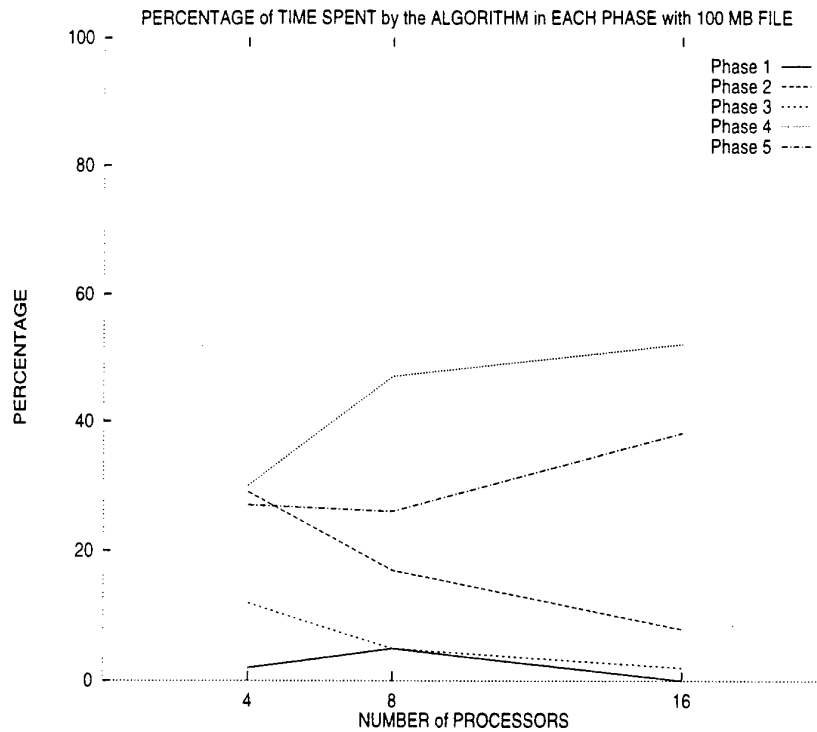


Fig. 1. Percentage of time spent by each step of the program, when it was submitted to a network of workstations in CENAPAD - MG/CO with a text input file of 100 MB.

66 MHz and 64 MB of main memory. The network has a switch at 40 MB of unidirectional bandwidth and a local disk for each processor.

The parallel program is written in ANSI C using MPI (Message Passing Interface) [4] as communication library. The benchmark textual database is composed of file texts of 100 MB and 200 MB, extracted from the Wall Street Journal of TREC-3 collection [5].

The experiment results presented in this article were obtained by executing the program in CENAPAD - MG/CO considering 1, 4, 8, and 16 processors. Some details should be stated:

- only 54 MB of main memory were used by the processors of CENAPAD - MG/CO. This limitation was set because it is the maximum portion of main memory used by the LMC processors, when experiments are performed in France. This memory size compatibility have to be kept because memory is important in this study;
- experiments with 2 processors were not performed due to the politic of memory utilization adopted by the program. By this politic, $\frac{2}{3}$ of main memory

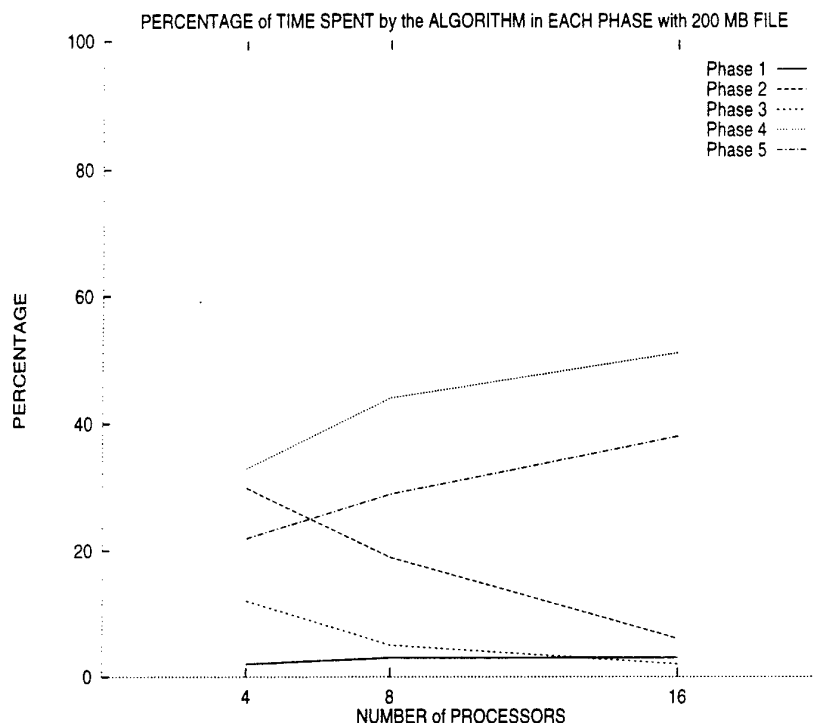


Fig. 2. Percentage of time spent by each step of the program, when it was submitted to a network of workstations in CENAPAD - MG/CO with a text input file of 200 MB.

is left to the PAT array and the other $\frac{1}{3}$ is left to the text. Two processor would have too many I/O disk operations, what would be a similar case of the sequential algorithm implementation.

Figures 3 and 4 present the performance of the program, considering text files of 100 MB and 200 MB. The speedup was measured considering an sequential implementation of the algorithm presented in [6].

It can be noticed (figure 4) that with 4 processors speedup is super-linear (5.12 and 5.76 with files of 100 MB and 200 MB, respectively) and with 16 processors speedup is bad. Super-linear speedup is observed because the implementation of the sequential algorithm has a quadratic behaviour due to I/O disk operations, although its complexity is $n \log n$ [6]. Sub-linear speedup occurs because of the single disk shared by every processors in CENAPAD - MG/CO. The more processors involved in the index generation, the greater is the number of files that each processor must deal with. These files are created by the processors during the broadcast (step 4). The competition by I/O bus and seek time of the disk determine the low performance of the program, in this envi-

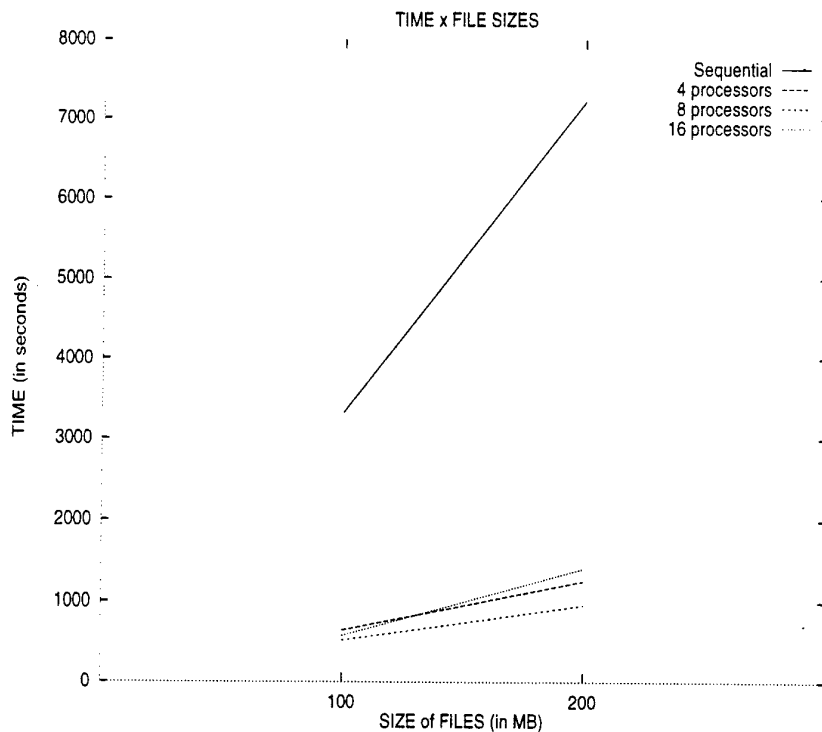


Fig. 3. Measures of elapsed time for index generation using 1, 4, 8 and 16 processors, connected by fast network at CENAPAD - MG/CO, considering files of different sizes.

ronment. The domination of I/O time in the performance of program can be observed in figures 5 and 6. The curves show the percentage of time spent in step 4 of the algorithm. It can be devised that the percentage of time in I/O operations is increasing, as increases the number of processors. There are 3 curves: communication, I/O operations, and others. This last curve reflects activities like:

- suffix array compression;
- package of data to be transmitted;
- contention in the suffix array transmission.

Figures 5 and 6 present also a turning point on curve "others", when the number of processors is 8. This turning point is caused by the competition of the processors for the disk. Besides, the graphics show the percentage of time spent by the algorithm when communication, I/O disk operations, and other factors are considered. Beyond 8 processors, the percentage of time of other factors (curve "others") decrease, on the hand the percentage of time of I/O disk operations increase.

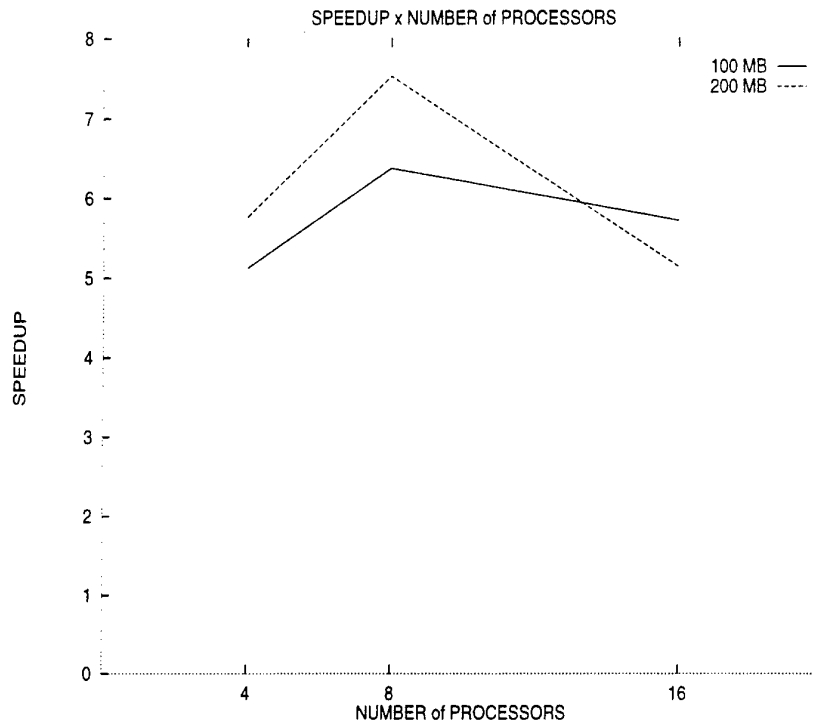


Fig. 4. Curves of speedup obtained in a network of workstations at CENAPAD - MG/CO.

3.2 Influence of Text Characteristics

One concern with the algorithm here presented is the load balance during steps 4 and 5. More specifically, in the step 4, a non homogeneous communication load occurs if exchanged *parts* of slices are of different sizes. In step 5 also, the resulting slice to be sorted locally would have different sizes for different processors. This would imply a non balanced computational load. This happens when the word frequency distribution is not *auto-similar*.

A structure is said *strictly auto-similar* [7] if it can be recursively decomposed in small pieces where each one is a replica of the original structure. It is important to say that these parts are obtained through a *scale transformation* of the original structure. Those structures that can be decomposed in similar parts *until a given scale* is said *auto-similar*.

In order to detect auto-similarity, experiments were done over the following collections [5] *AP* (Associated Press) (1988), *WSJ* (1987), and *Ziff-Davis* (complete). The collection files were split into pieces of 1,000; 10,000; 100,000;

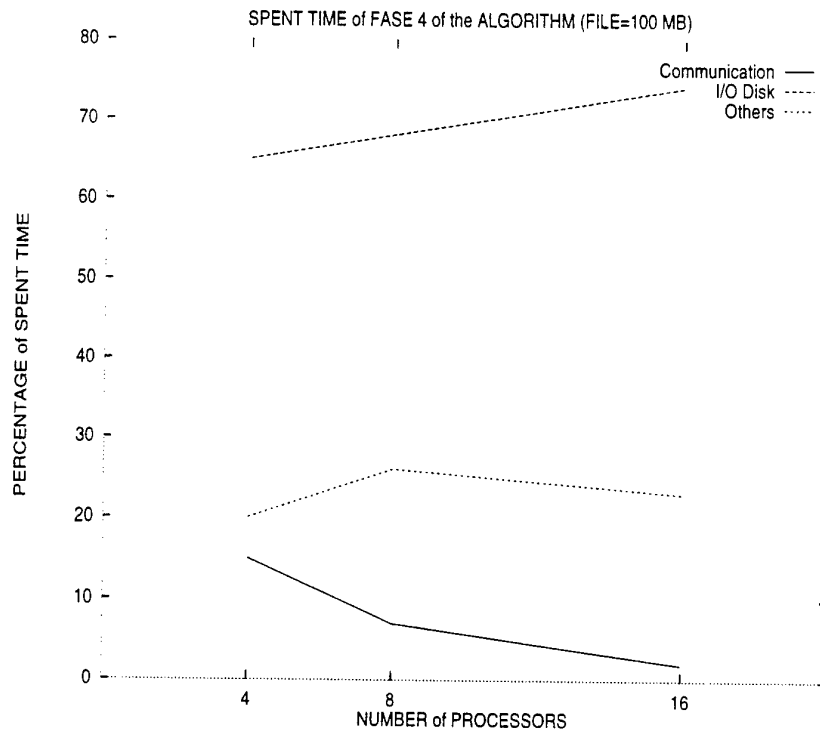


Fig. 5. Percentage of time spent in step 4 of the algorithm (which includes communication, I/O operations on disk, compression of suffix arrays, packaging of data to be transmitted, contention in suffix array transmission), considering a file of 100 MB.

300,000; 550,000; 700,000; 850,000; and 1,000,000 words. These different sizes were to detect if the number of words in files of different sizes grows linearly.

Figures 7 and 8 describe the number of words starting with a given letter *versus* the alphabet letters, independently of the text file size. Since these graphics have the same shape, they are similar, from the geometric point of view. This similarity occurred in all files collections experienced.

Table 1 presents linear regression information concerning each alphabet letter for the AP collection. The numbers in this table confirm that the number of words starting by a given letter ℓ grows linearly with the text file size. The coefficient of determination of the linear regression is close to 1 (see third column). Similar results were obtained in Wall-Street Journal and Ziff-Davis Collection.

The second column of Table 1 (times 100) can be also considered as the percentage of words starting with a given letter ℓ in a text file of size t . Ignoring round errors, the total sum of the numbers of the second column of this table is 1.

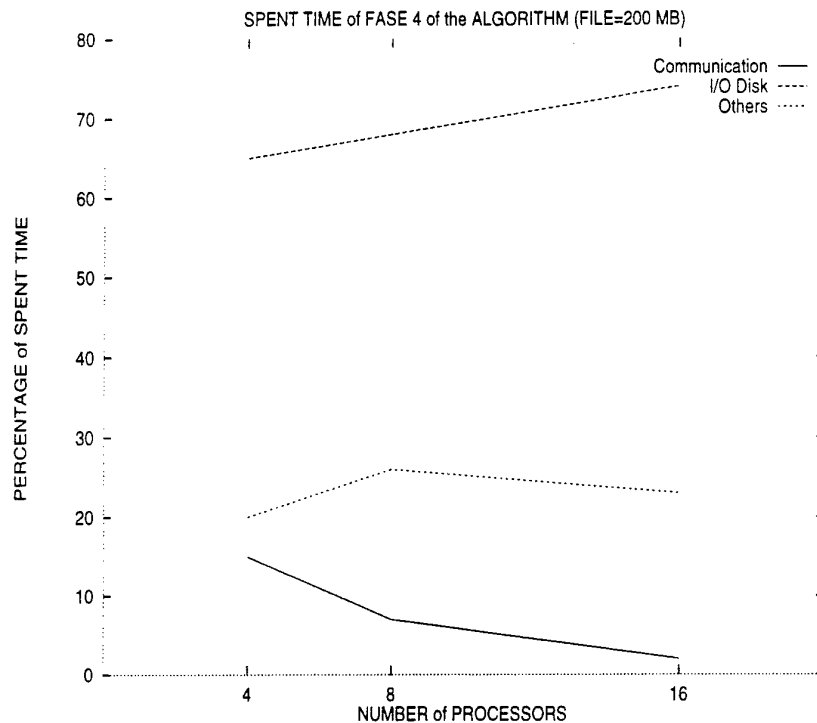


Fig. 6. Percentage of time spent in step 4 of the algorithm (which includes communication, I/O operations on disk, compression of suffix arrays, packaging of data to be transmitted, contention in suffix array transmission), considering a file of 200 MB.

On a probabilistic point of view, Table 2 presents the distribution function of the word frequency. Due to the auto-similarity, this function is independent of the text file size. For example, the probability of choosing a word starting by letters A or B or ... or K is around 50%, independent of the file size.

Experimentally, the text auto-similar feature shows that the computed global percentiles effectively generates a homogeneous distribution of suffix pointers among the processors (step 4: communication load balance). The number of bytes sent and received by each processor is almost the same for all processors. Consequently, the final local sort (step 4) will work with roughly the same number of pointers, implying a computational load balance. The same conclusion was obtained by simulation of the algorithm [2].

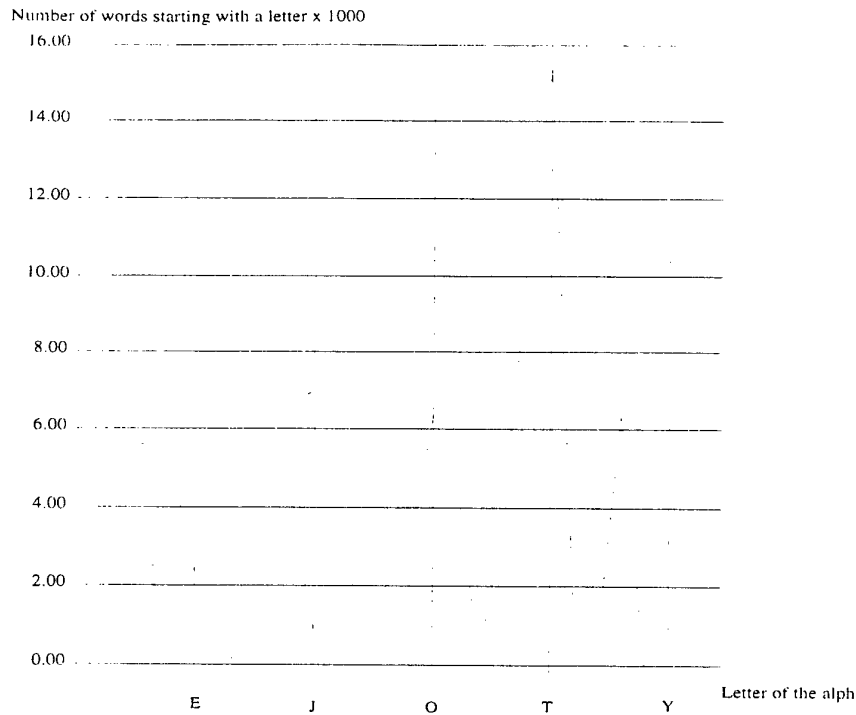


Fig. 7. Distribution of the word frequency (considering just the first letter) in the AP collection, with 100,000 words.

4 Conclusions

This paper presented experiments performed with an implementation of a quick-sort-based parallel indexing algorithm. Besides the expected reduction in execution time, it was observed that the word frequency distribution of the input textual database has an significant influence on performance.

The experiments were performed in CENAPAD -MG/CO, a supercomputing center whose machine (an IBM SP) share a single disk among all processors. This computation environment implied in a performance decrease of the program as the number of processors (involved in the index generation) is greater or equal 16. This distortion occurred due to the great competition of processors by I/O bus and the seek time of the disk.

As future work, experiments will be performed with large files and launched in LMC (France). In LMC there is a disk for each processor and this fact will certainly decrease of the I/O bus competition and it will be possible to see better curves of speedup.

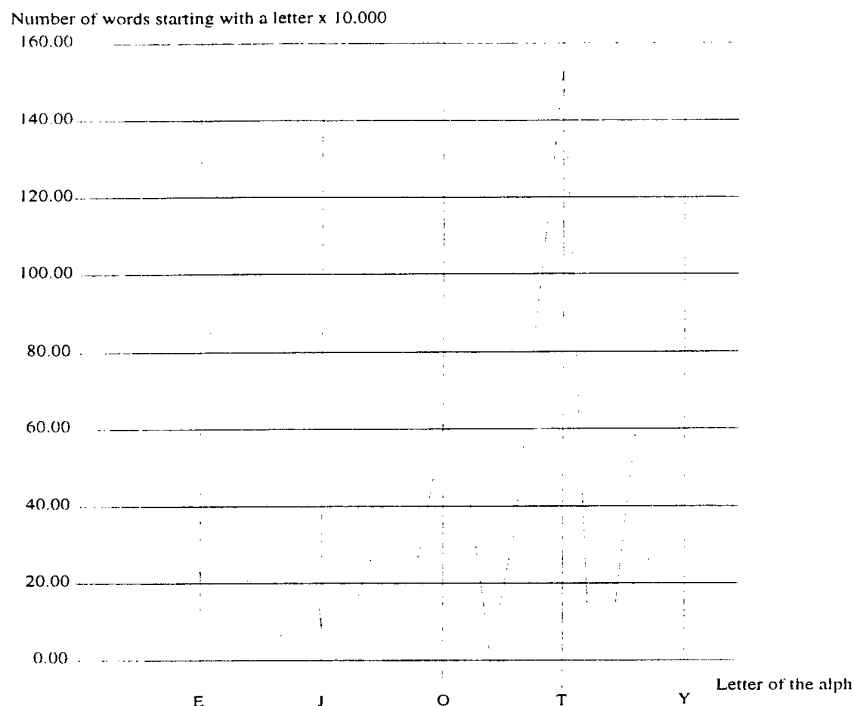


Fig. 8. Distribution of the word frequency (considering just the first letter) in the AP collection, with 1,000,000 words.

References

1. R. Frakes and R. Baeza-Yates. *Information Retrieval: Algorithms and Data Structures*. Prentice-Hall, 1992.
2. G. Navarro, J. P. Kitajima, B. A. Ribeiro-Neto, and N. Ziviani. Distributed generation of suffix arrays. In A. Apostolico and J. Hein, editors, *Lecture Notes in Computer Science*, volume 1264, pages 102-115, Aarhus, Denmark, June 1997. Springer-Verlag. Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching (CPM).
3. T. Anderson, D. Culler, and D. Patterson. A case for NOW (Network Of Workstations). *IEEE Micro*, 15(1):54-64, 1995.
4. IBM. *IBM Parallel Environment for AIX: MPI Programming and Subroutine*, 1997. Version 2, Release 3, Doc. Number GC23-3894-02.
5. D. Harman. Overview of the 3rd Text REtrieval Conference (TREC-3). In *Proceedings of the 3rd Text REtrieval Conference (TREC-3)*, 1995.
6. Gaston H. Gonnet, Ricardo Baeza-Yates, and T. Snider. *New indices for text: PAT trees and PAT arrays*, page Chapter 5. 1992.
7. H. O. Peitgen, H. Jürgens, and D. Saupe. *Fractals for the Classroom - Part One: Fractals and Chaos*. Springer-Verlag, 1992.

ℓ	Slope	Coeff. of Determ. (R^2)
A	0,123193195	0,997446196
B	0,05259202	0,997838698
C	0,055195342	0,998992098
D	0,046114325	0,998172851
E	0,021607113	0,997351481
F	0,05154366	0,99805157
G	0,017667218	0,998051994
H	0,047498466	0,996058323
I	0,061164202	0,997900196
J	0,008406008	0,99654862
K	0,005621952	0,992414744
L	0,024723614	0,998464284
M	0,0386796	0,997983352
N	0,026271595	0,998471551
O	0,059174769	0,997932797
P	0,047215617	0,99792522
Q	0,001560132	0,99621972
R	0,033622413	0,997980403
S	0,093770112	0,997193251
T	0,14873902	0,998472823
U	0,014391688	0,999281988
V	0,006702519	0,998155787
W	0,0573626	0,997223945
X	0,00012865	0,968463675
Y	0,008135312	0,997146319
Z	0,000446293	0,979992358

Table 1. Slope and coefficient of determination obtained from linear regression for words starting by ℓ (regression of the measures of the text file size *versus* word frequency starting by a given letter ℓ).

Interval	Percentage of Text
A	0,123193195
A-B	0,175785215
A-C	0,230980556
A-D	0,277094881
A-E	0,298701995
A-F	0,350245655
A-G	0,367912872
A-H	0,415411338
A-I	0,47657554
A-J	0,484981548
A-K	0,4906035
A-L	0,515327114
A-M	0,554006714
A-N	0,580278309
A-O	0,639453078
A-P	0,686668695
A-Q	0,688228827
A-R	0,72185124
A-S	0,815621353
A-T	0,964360373
A-U	0,978752061
A-V	0,985454579
A-W	1,042817179
A-X	1,042945829
A-Z	1,051081141

Table 2. Distribution function of the word frequency per letter ℓ .

A Low Cost Distributed System for FEM Parallel Structural Analysis

Célio Oda Moretti¹, Túlio Nogueira Bittencourt¹, and Luiz Fernando Martha²

¹ Computational Mechanics Laboratory, Department of Structural and Foundation Engineering, Polytechnic School, University of São Paulo,
Av. Prof. Almeida Prado, travessa 2, No. 271,
CEP 05508-900 - São Paulo - Brazil
{moretti, tbitten}@usp.br
<http://www.lmc.ep.usp.br>

² Department of Civil Engineering and Technology Group on Computer Graphics - TeCGraf, Pontifical Catholic University of Rio de Janeiro - PUC-Rio,
Rua Marquês de São Vicente, No. 225,
CEP 22453-900 - Rio de Janeiro - Brazil
lfn@tecgraf.puc-rio.br
<http://www.tecgraf.puc-rio.br>

Abstract. In this paper, a distributed computational system for finite element structural analysis and some strategies for improving its efficiency are described. The system consists of a set of programs that performs the structural analysis in a distributed computer network environment. This set is composed by a pre-processor, a post-processor, a program responsible for partitioning the model in substructures, and by a structural analysis parallel solver. The domain partitioning is performed interactively by the user through a graphics interface program (PARTDOM). An existing FEM code, based on object oriented programming concepts (FEMOOP), has been adapted to implement the parallel features. Different implementation aspects concerning scalability and performance speed-up are discussed. The computational environment consists of a 100 Mbit Fast-Ethernet network cluster including eight Pentium 200 MHz micro-computers running under LINUX operating system.

1 Introduction

In this work, a distributed computational system for finite element structural analysis will be presented. This system has been developed to perform the parallel analysis using a disposable local area network as the processing environment. This arrangement makes possible the use of low cost parallel capabilities, avoiding the utilization of expensive supercomputers. A set of integrated components, each one responsible for a specific task in the analysis process, comprises the computational system. The four basic units of this system are presented in Fig. 1.

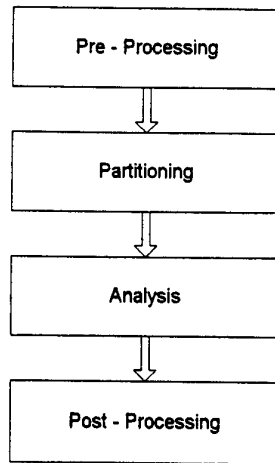


Fig. 1. Basic units of the parallel analysis system

This set has four components: a pre-processor, a post-processor, a program responsible for partitioning the model in substructures, and a structural analysis parallel solver. The main focus here is the domain partitioning and the solver, in which the parallel features have been implemented. The domain partitioning is performed interactively by the user through a graphic interface program, called PARTDOM. Three automatic substructuring procedures are available and the user is able to manually edit the resulting subdomain partitions. An existing FEM code, based on object oriented programming concepts (FEMOOP), has been adapted to implement the parallel features. The main extensions to the code are the introduction of a preconditioned conjugate-gradient parallel algorithm to solve the global equations and the communication procedures among the processors.

In the following sections, the parallel analysis system and its components will be described in detail. Also, different aspects affecting the system performance will be discussed. The performance is measured through the time consumed in each step of the analysis processing for different number of processors. Some strategies for improving the system efficiency have been implemented and their results will be discussed.

2 The Parallel Analysis System

All components of this parallel system are presented in this section. Special attention is placed on the domain partitioning and the analysis programs. The main development and implementations are located in these programs.

2.1 Pre-Processing

The package MTOOL (Bidimensional Mesh Tool) [1] has been used as the pre-processor in this work. MTOOL is an interactive graphics program for bidimensional finite element mesh generation. With this program, the geometry, material properties, and boundary conditions of the model are defined. Through its graphical interface (Fig. 2), the program makes the visualization and the editing of the generated mesh possible.

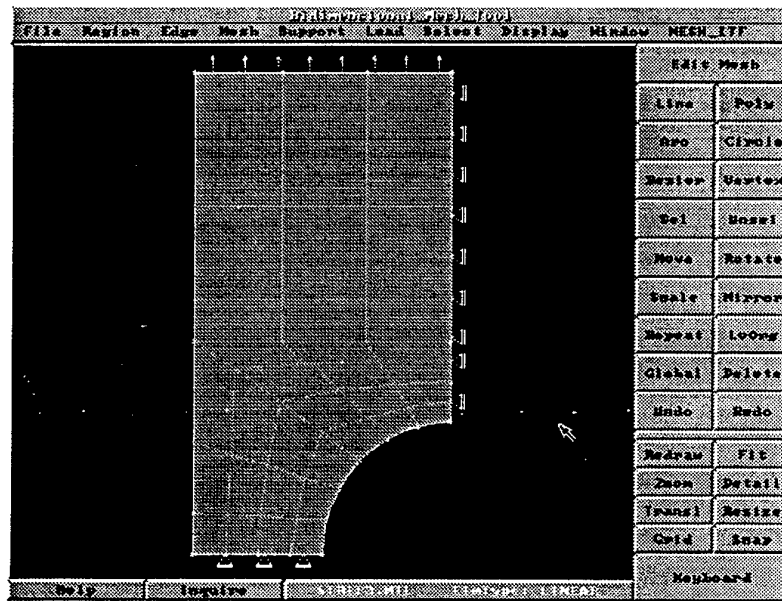


Fig. 2. MTOOL graphical interface

At the end of model creation, a neutral format file is generated. This file contains, in a standard way, all the information about the model, which are necessary in the analysis process. The neutral file is an essential feature for the integration of all components of the parallel system.

2.2 Partitioning

After the model generation, the structure has to be partitioned into a number of substructures to take advantage of the parallel environment. This partitioning is made through the use of automatic domain partitioning algorithms. The main objectives of these algorithms are: (1) to obtain a balanced work distribution among the processors, and (2) to minimize the boundary degrees of freedom (DOF) present between the

substructures. These two objectives represent important aspects that affect the overall system performance.

The basic aim of PARTDOM program[2] is to facilitate the user interaction with the automatic domain partitioning algorithms. The program allows the partition of a bidimensional finite element mesh using three different algorithms. Through its graphical interface (Fig. 3), the program permits the visualization and manipulation of the resulting subdomain partitions. Thus, PARTDOM has been developed with the following objectives:

- ☐ to promote user interaction with the automatic domain partitioning algorithms,
- ☐ to facilitate partitioning of complex meshes, through a graphical interface that allows visualization of resulting partitions,
- ☐ to allow interactive user edition of resulting partitions,
- ☐ to be portable, i.e., the program code is platform independent.

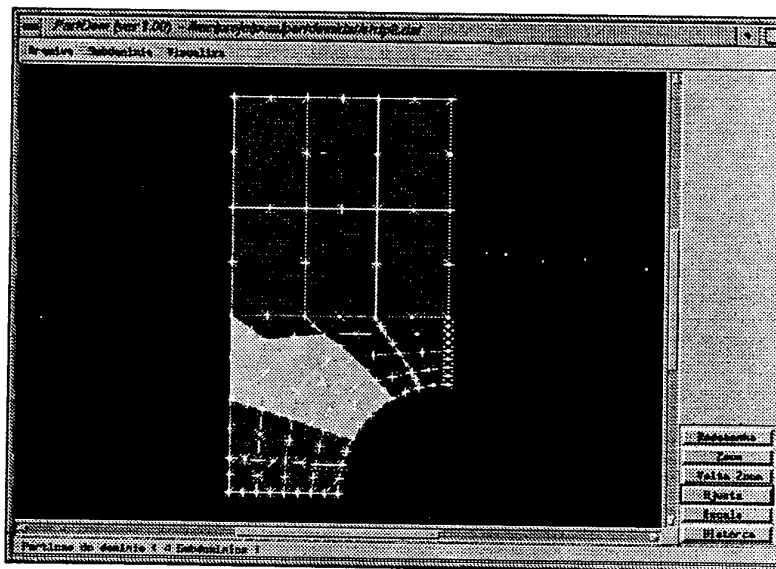


Fig. 3. PARTDOM graphical interface

The three partitioning algorithms used in PARTDOM are: Al-Nasra & Nguyen algorithm [3] and two algorithms from the METIS library [4] called *pmetis* and *kmetis*. Fig. 4 presents the resulting partitions applying the three algorithms over a finite element mesh composed by 587 16 elements.

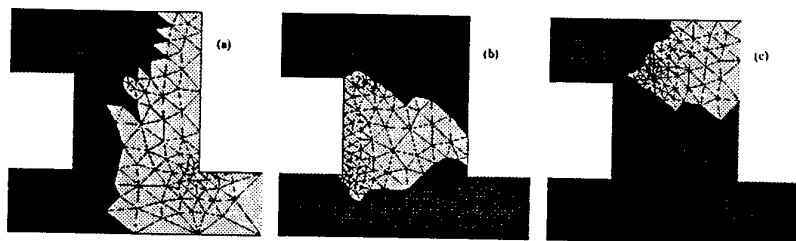


Fig. 4. Mesh partitions resulting from the use of algorithms: (a) Al-Nasra & Nguyen, (b) *pmetis*, and (c) *kmetis*

The three algorithms present a good balanced work distribution among the processors, with approximately the same number of elements for each substructure. However, the Al-Nasra & Nguyen algorithm generally presents a number of boundary DOF between the substructures greater than the *pmetis* and *kmetis* algorithms. This characteristic increases the requirement of communication between the processors, and, consequently, decreases the parallel analysis performance. The partitions resulting from the use of *pmetis* and *kmetis* algorithms present approximately the same number of boundary DOF.

At the end of this process, the partitioning information is added to the neutral file.

2.3 Analysis

After the model generation and partitioning, the next step consists of the parallel analysis of this model, with the use of a set of processors, each of one responsible for a part of the computational work. In this work, the Finite Element Method has been employed in the analysis. A analysis program named FEMOOP (Finite Element Method - Object-oriented Programming)[5], developed at the Department of Civil Engineering (PUC-Rio) and at Computational Mechanics Laboratory (Polytechnic School / USP), has been used as the platform for the new parallel capabilities additions.

The program FEMOOP is organized using object-oriented concepts of the C++ programming language[6][7]. One of the most important advantages of the object-oriented programming is the code extensibility. This feature allows new implementations with minimum impact over the existent code. To adapt FEMOOP to the parallel computational environment a new class has been created, which is responsible for data manipulation. Also, a series of new functions has been implemented into existent classes.

The message-passing manager used in this work has been PVM (Parallel Virtual Machine)[8]. PVM is a software system that permits a heterogeneous computer network to be viewed as a single parallel computer. The first step necessary to adapt FEMOOP to the parallel environment has been the implementation of a library responsible for the message passing management. The main objective of this library is to limit the direct access to PVM functions. An eventual change of the message-passing manager is facilitated, which has impact only over the library code. This

parallel procedure library contains all functions necessary to perform the message passing in a distributed memory environment. The main functions implemented here are responsible for sending and receiving messages among processors, for parallel process initialization, and for the identification of program type (if it is either a master or a task program).

The parallel programming paradigm adopted here has been the master-slave model. In this model, the master is a separate program responsible for process spawning, initialization, reception and display of results, and timing of functions. The task (or slave) programs are executed concurrently and interact through message passing. The actual structural analysis is done by the task programs, each of one responsible for the work corresponding to one substructure. Through interaction between these task programs, the global solution is obtained and then it is sent to the master program.

To obtain the linear system of equilibrium equations a substructuring technique[9] has been employed. When this technique is used, the original structure is partitioned into a number of substructures, which are distributed among the processors. The substructure degrees of freedom are classified as internal DOF and boundary DOF, which are shared between neighbor substructures. The substructures interact through these boundary DOF only. Then, the stiffness matrices of each substructure are mounted. In this work, the internal unknowns are eliminated using Crout method. After this step, a condensed system with terms corresponding only to boundary unknowns is obtained. All these procedures can be performed concurrently. To solve the partitioned global system, a parallel iterative solver has been used. A parallel implementation of the pre-conditioned conjugate gradient (PCG) method[10][9] has been chosen as the solution method adopted in this work. Basically, this parallel implementation of the PCG method consists of parallel operations between matrices and vectors. The sequence of operations is the same both in the parallel and in the sequential versions of the PCG method.

To employ this substructuring technique, the stiffness matrix $K^{(i)}$, the force vector $f^{(i)}$, and the nodal unknowns $u^{(i)}$, corresponding to each substructure i , have been mounted with terms partitioned into internal and boundary terms. These partitions are presented in (1), where indices I and S correspond respectively to internal and boundary (or shared) terms.

$$K^{(i)} = \begin{bmatrix} K_I^{(i)} & K_{IS}^{(i)} \\ K_{IS}^{(i)r} & K_S^{(i)} \end{bmatrix}, f^{(i)} = \begin{bmatrix} f_I^{(i)} \\ f_S^{(i)} \end{bmatrix}, u^{(i)} = \begin{bmatrix} u_I^{(i)} \\ u_S^{(i)} \end{bmatrix}. \quad (1)$$

For each substructure, the linear equation system is written in the form

$$\begin{bmatrix} K_I^{(i)} & K_{IS}^{(i)} \\ K_{IS}^{(i)r} & K_S^{(i)} \end{bmatrix} \begin{bmatrix} u_I^{(i)} \\ u_S^{(i)} \end{bmatrix} = \begin{bmatrix} f_I^{(i)} \\ f_S^{(i)} \end{bmatrix}. \quad (2)$$

Eliminating the internal unknowns, a condensed equation system is obtained

$$\bar{K}_S^{(i)} u_S^{(i)} = \bar{f}_S^{(i)}, \quad (3)$$

where $\bar{K}_s^{(i)}$ and $\bar{f}_s^{(i)}$ are respectively the condensed stiffness matrix and the condensed force vector, and

$$\bar{K}_s^{(i)} = K_s^{(i)} - K_{IS}^{(i)T} K_I^{(i)-1} K_{IS}^{(i)}, \quad (4)$$

$$\bar{f}_s^{(i)} = f_s^{(i)} - K_{IS}^{(i)T} K_I^{(i)-1} f_I^{(i)}. \quad (5)$$

The global linear equation system can be written in the form

$$\left[\sum_{i=1}^p L^{(i)T} \bar{K}_s^{(i)} L^{(i)} \right] = \sum_{i=1}^p L^{(i)T} \bar{f}_s^{(i)}, \quad (6)$$

where p is the number of substructures and $L^{(i)}$ is a boolean matrix that describes the substructure connectivity in the original structure.

2.4 Post-Processing

A program named MVIEW (Bidimensional Mesh View)[11] has been utilized as the post-processor tool. MVIEW is an interactive graphical program (Fig. 5) for visualization of structural analysis results. This program provides visualization of the deformed configuration and contours of scalar results at nodes and Gauss points.



Fig. 5. MVIEW graphical interface

3 Aspects of Parallel Processing

In this section, different aspects that affect the parallel analysis system performance are presented.

The quality of partition obtained from application of automatic domain partitioning algorithms affects the global performance of the parallel system. A good partition generates substructures with approximately the same number of elements, and with minimum number of boundary nodes between neighbor substructures. The equilibrated distribution of elements is important to balance the computational work load among the processors, avoiding occurrence of idle periods. Idle periods occur when a processor interrupts its job to wait for a information from a busy processor. These waiting intervals cause system performance degradation. The number of boundary nodes between substructures is a important aspect that influences the parallel system solver performance. When the number of boundary nodes increases, the number and size of messages exchanged during the solution phase also increase. The three automatic domain partitioning algorithms used in this work present a good work load distribution, but Al-Nasra & Nguyen algorithm usually produces boundaries with a greater number of nodes than *pmetis* and *kmetis* algorithms[12].

The message passing also influences the initialization process executed by the master program. In this analysis phase, the master program spawns task programs and sends all information needed by tasks to perform the analysis. This information includes geometry data, boundary conditions, material properties, etc. When the model size increases, the message sizes also increase, reducing consequently the system performance.

The way the condensed stiffness matrix (equation (4)) is mounted is the most important aspect that affects the parallel analysis system performance. First, system equation (7) is solved

$$K_I^{(i)} w = k_{IS}^{(i)}, \quad (7)$$

where $k_{IS}^{(i)}$ is a column of $K_{IS}^{(i)}$, and the operation (8) is carried out to obtain a line $\bar{k}_S^{(i)}$ of $\bar{K}_S^{(i)}$

$$\bar{k}_S^{(i)} = k_S^{(i)} - w^T K_{IS}^{(i)}. \quad (8)$$

Therefore, solving equations (7) and (8) to each column of $K_{IS}^{(i)}$, all lines of the condensed stiffness matrix are obtained. This process is highly influenced by the number of internal DOF of the substructure and by the bandwidth of the stiffness matrix $K_I^{(i)}$ since this matrix must be decomposed to solve equation (7).

4 System Performance

The parallel system performance is presented in this section. Two versions of this system have been evaluated and compared for the same model. The second version incorporates implementations to improve the system performance.

The hardware setup consists of a 100 Mbit Fast-Ethernet network cluster including eight Pentium 200 MHz micro-computers running under LINUX operating system. This cluster is fully dedicated to parallel processing.

The numerical example used to measure system performance is a beam with geometry, boundary conditions and mesh presented in Fig. 6. The model attributes are: $E = 7000 \text{ kN/cm}^2$, $\nu = 0.25$ and thickness = 1 cm.

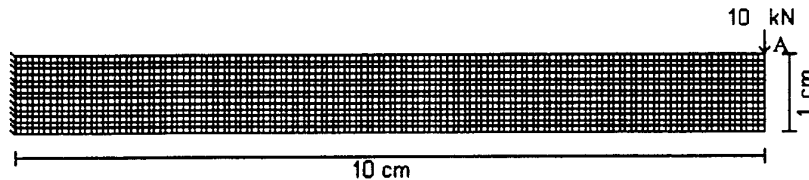


Fig. 6. Model geometry, boundary conditions and mesh

The model is created and discretized using a regular mesh of 13×130 plane stress Q8 elements. The mesh is partitioned into 2 to 8 substructures, using the *kmetis* algorithm available in PARTDOM. Fig. 7 presents the resulting mesh partitioning when 4 substructures are used.



Fig. 7. Mesh partitioning into 4 substructures

Through the analysis of performance of the first version of the parallel system, the main bottleneck aspects have been identified. These aspects have been presented in the last section. The second version of the parallel system incorporates new implementations to improve the performance of critical system routines.

In the first FEMOOP parallel version, the master program is responsible for initialization, spawning of task programs, and sending all information needed by tasks to perform the actual analysis. This information include all geometry data, topological data, material properties, boundary conditions, and domain partitioning data. With a large scale model, this time step becomes significant. In the second version, the task programs read information directly from data files, avoiding large message exchanging.

Another aspect influencing system performance is the decomposition of $K_I^{(i)}$ matrix, required to mount the condensed stiffness matrix. Usually this step causes an unbalanced computational work distribution among processors because the substructures, generated through automatic domain partitioning algorithms, present a stiffness matrix that is not optimized to numerical analysis. Thus, in the second version, FEMOOP is able to perform nodal reordering for each substructure, reducing the time to mount the condensed stiffness matrix.

In the new FEMOOP version, the pointers to nodal objects are obtained through a vector request, and not through a linked list of pointers. The use of a vector of pointers increases the overall system performance.

The main parallel analysis phases are presented in Fig. 8, 9, 10 and 11. These graphics compare the phase time consuming between first and second FEMOOP versions. Fig. 8 presents time consumed to send substructure information to each processor. Fig. 9 presents time consumed to mount the condensed stiffness matrix. Fig. 10 presents time needed to solve the linear system of equations using the parallel solver. And in Fig. 11 the time consumed to perform complete analysis is depicted.

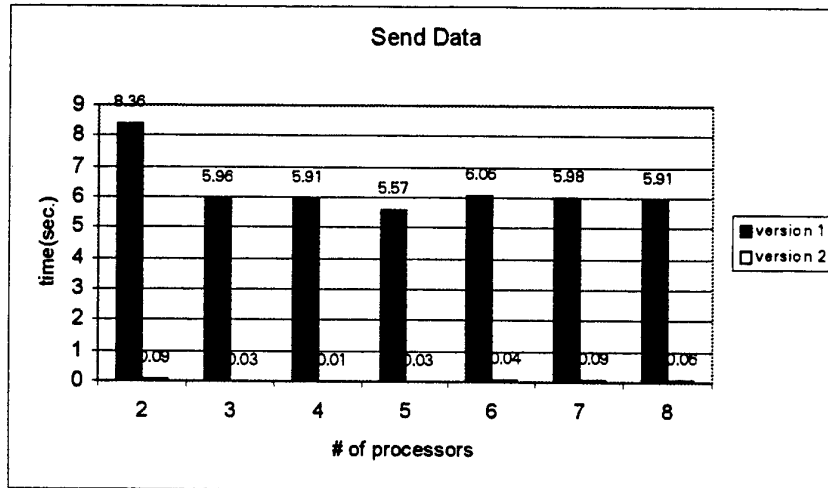


Fig. 8. Time consumed to send substructure data to processors

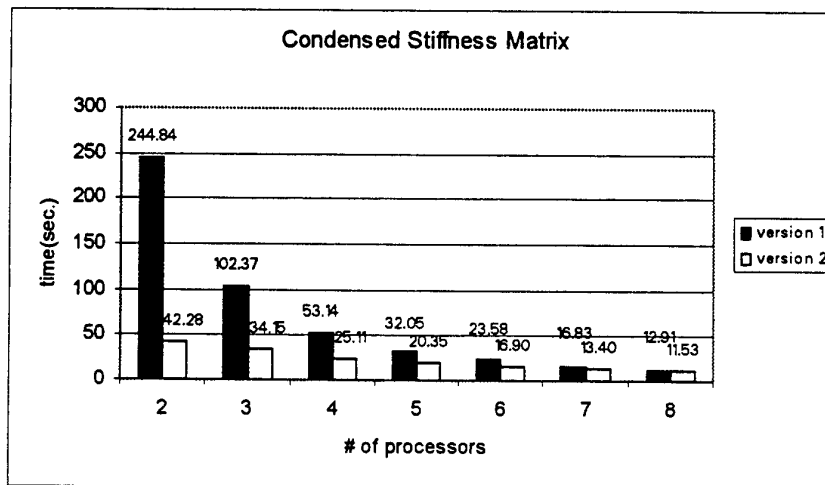


Fig. 9. Time consumed to mount condensed stiffness matrix

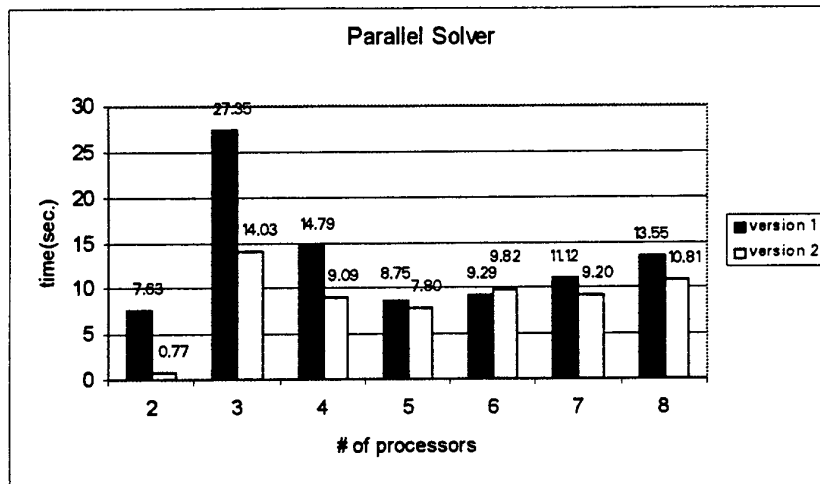


Fig. 10. Time consumed to solve linear equation system using the parallel solver

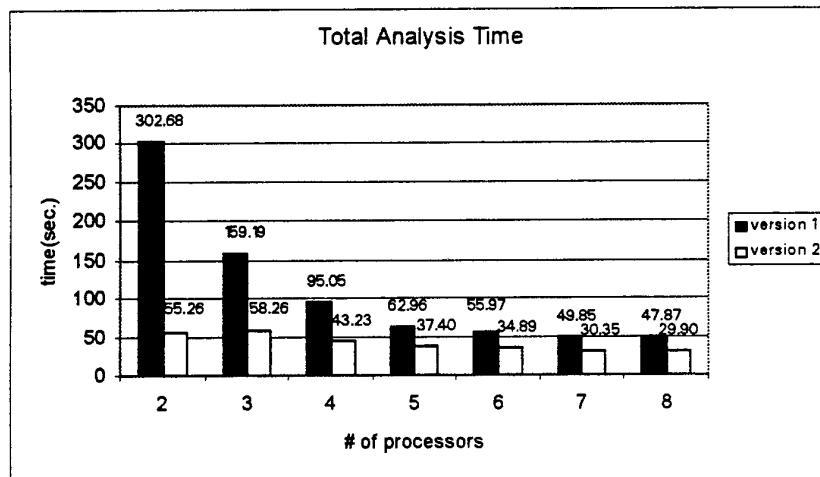


Fig. 11. Total time consumed to complete the analysis

The effect of direct access to data files is evident (Fig. 8). The time consumed to perform the data transmission is much lower in the second FEMOOP version. The disadvantage of direct access is that an up-to-date data file must be available to each system processor.

The nodal reordering has a crucial effect on the condensed stiffness matrix mounting routine. Fig. 9 shows a great time reduction in the second FEMOOP version, specially for a small number of substructures. To mount the condensed stiffness matrix, decomposition of $K_I^{(i)}$ matrix is needed. The time consumed in this decomposition also decreases greatly with nodal reordering.

The parallel solver is highly affected by synchronization problems (Fig. 10). The first step of the parallel solver is the mounting of pre-conditioning matrix. To accomplish this, message passing is needed. Thus, the time consumed by the solver incorporates idle periods of processors waiting for information from another busy processor. This is a problem that appears in both FEMOOP versions. Nodal reordering has not been sufficient to address this problem.

The new implementations allowed a reduction in the total analysis time (Fig. 11) when 2 to 8 processors are used simultaneously.

5 Conclusions

A parallel structural analysis system has been presented in this work. This parallel system permits a better utilization of the resources present in a local area network. This arrangement enables a low cost parallel structural analysis environment, avoiding for certain classes of problems the utilization of expensive supercomputers.

A first version of the parallel system has been used to identify the main aspects affecting the global system performance. New implementations have been added to the code to improve performance. The development of this system takes advantage of the object-oriented programming concepts, which permits new implementations without great impact over the existing code.

The nodal reordering and the direct access to data files improved greatly the system performance. The reduction of the time consumed to mount the condensed stiffness matrix and to send data over to all processors confirm the adequacy of these new implementations.

Some improvements are still needed to resolve the synchronization problem that appears in the current system. One of these improvements is certainly a better work distribution among processors.

This system will be used in a future work to perform 3D fracture analysis of cracked structures. To accomplish this objective, new features and improvements will be necessary. Automatic 3D domain partitioning and dynamic computational work distribution are some of these improvements.

References

1. MTOOL - Bidimensional Mesh Tool (Versão 1.0) - Manual do Usuário, Grupo de Tecnologia em Computação Gráfica - TeCGraf / PUC-Rio, 1992.
2. Moretti, C.O., Bittencourt, T.N., André, J.C., and Martha, L.F., "Algoritmos Automáticos de Partição de Domínio", Boletim Técnico, Departamento de Engenharia de Estruturas e Fundações, Escola Politécnica - USP, 1998 (a ser publicado).
3. Al-Nasra, M. e Nguyen, D.T., "An Algorithm for Domain Decomposition in Finite Element Analysis", Computers & Structures, Vol. 39, No. 3/4, pp. 277-289, 1991.
4. Karypis, G. e Kumar, V., "METIS: Unstructured Graph Partitioning and Sparse Matrix Ordering", Department of Computer Science, University of Minnesota, 1995.

5. Martha, L.F., Menezes, I.F.M., Lages, E.N., Parente Jr, E. and Pitangueira, R.L.S., "An OOP Class Organization for Materially Nonlinear Finite Element Analysis", Join Conference of Italian Group of Computational Mechanics and Ibero-Latin American Association of Computational Methods in Engineering, pp. 229-232, University of Padova, Padova, Italy, 1996.
6. Fujii, G., "Análise de Estruturas Tridimensionais: Desenvolvimento de uma Ferramenta Computacional Orientada para Objetos", Dissertação de Mestrado, Dep. de Engenharia de Estruturas e Fundações (PEF), Escola Politécnica, USP, 1997.
7. Guimarães, L.G.S., Menezes, I.F.M. and Martha, L.F., "Object Oriented Programming Discipline for Finite Element Analysis Systems" (in Portuguese), Proceedings of XIII CILAMCE, Porto Alegre, RS, Brasil, Vol. 1, pp. 342-351, 1992.
8. Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., Sunderman, V., *PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, 1994.
9. Nour-Omid, B., Raefsky, A., e Lyzenga, G., "Solving Finite Element Equations on Concurrent Computers", in A.K. Noor, Ed., *Parallel Computations and Their Impact on Mechanics*, pp. 209-227, ASME, New York, 1987.
10. Hestenes, M. e Stiefel, E., "Methods of Conjugate Gradients for Solving Linear Systems", *Journal of Research of the National Bureau of Standards*, Vol. 49, No. 6, pp. 409-436, Research Paper 2379, December 1952.
11. MVIEW - *Bidimensional Mesh View* (Versão 1.1) - Manual do Usuário, Grupo de Tecnologia em Computação Gráfica - TeCGraf / PUC-Rio, 1993.
12. Moretti, C.O., "Análise de Estruturas Utilizando Técnicas de Processamento Paralelo Distribuído", Dissertação de Mestrado, Dep. de Engenharia de Estruturas e Fundações (PEF), Escola Politécnica, USP, 1997.

Low cost parallelizing A way to be efficient.

Marc Martin and Bastien Chopard

CUI, University of Geneva
24 rue General-Dufour,
CH-1211 Geneva 4, Switzerland

Keywords: Parallel computing; Dynamic load balancing; PVM;
Workstation cluster; Wave propagation; Pollution transport

Abstract

In order to minimize the heavy cost of running parallel applications on expensive MPPs, we consider the use of workstation clusters. The main difficulty to obtain efficiency on such architectures is the fact that a slow node may dramatically decrease the overall performance of the machine. We propose a dynamic load balancing solution, based on a local partitioning scheme, consisting of moving sub-domain boundaries so as to adjust the processor load according to its available CPU capability.

1 Introduction

Traditionally, heavy scientific applications are run on large and expensive mainframes such as massively parallel computers (MPPs) which are efficient but rather expensive. Price is not the single problem: in order to regulate the available CPU resources, system managers configure the computer for accepting large jobs only in batch mode. Then, even if the mainframe is very power-full, the user may wait for a long time before the job complete.

On the other hand, commodity computers like workstations or personnel computers become more and more powerful and cheaper. With an appropriate message passing library such as PVM, they represent an interesting alternative to MPPs in order to run heavy parallel applications in a fully interactive way and at better cost.

However, using such a solution is not as simple as porting the code from the parallel mainframe to the workstation cluster. A crucial problem of load balancing may appear because, usually, the processing nodes enrolled in a cluster

may not have all same amount of CPU resources. This may be due to a difference in the chip power, or because they are not running in a single-user mode. Another user may need the resources making the CPU balance very uneven.

Such unbalanced situations are very crucial problems because, in most cases, processing nodes must be synchronized from time to time in a parallel application. Since the global time process is lined up to the slowest node, one slow node is enough to reduce considerably the benefit of parallelization. Unfortunately, the more nodes we use, the more critical this situation will be. Thus, it is essential to implement algorithms handling load-unbalance problems.

2 Load balancing algorithm

Our dynamic load balancing algorithm is designed for a 2D regular grid divided along the north/south and east/west direction (see fig. 1). This domain decomposition leads to a set of rectangular sub-domains, each assigned to one PE. Rectangular shapes have the significant advantage that they make possible fast regular communications between adjacent processors, as opposed, for instance, to a cyclic partitioning.

Our algorithm determines each sub-domain size according to the work load of the PE. The sub-domain owned by an over-loaded PE will shrink while that of an unloaded PE will grow. A variation of size is obtained by moving sub-domain boundaries in a coherent way: when a sub-domain attempts to grow, the neighbors are forced to shrink, as illustrated in fig. 1. The problem is then to compute, in a minimal amount of time, the boundary motions which yield a fair work distribution.

The measure of load unbalance we consider here is based on the time each PE has spent during the last computation step. This measure, which has the advantage of being problem-independent, is appropriate in all applications where the same computation is iterated many times.

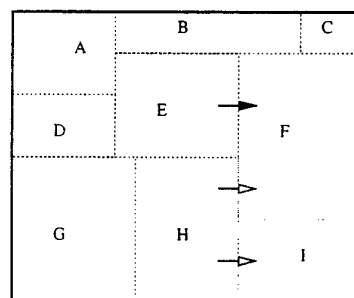


Fig. 1. Moving east boundary of node *E* will act on node *F* west boundary, and also on nodes *H* and *I* boundaries.

The load balancing problem can be addressed in two ways [KW] [Fos95]: on the one hand we can use a global policy which is very stable and accurate (because it accounts for every processor load), but may not be scalable. On the other hand, we can consider a local policy which may have a slow convergence and lack of stability, but which only requires local communications.

2.1 The global scheme

We consider n PEs involved in a parallel computation. At the end of step t , PE number i knows $T_{i,t}$, the time necessary to complete this step on its rectangular sub-domain of sizes $S_{i,t}$. Using an all-to-all communication, the values $T_{i,t}$ and $S_{i,t}$ are broadcast to all other PEs. From these values, each PE computes the ideal time Ta_t

$$Ta_t = \frac{1}{n} \sum_{i=1,n} T_{i,t} \quad (1)$$

and $Tc_{i,t}$ the time used by PE i for computing one single cells of domain $S_{i,t}$.

$$Tc_{i,t} = \frac{T_{i,t}}{S_{i,t}} \quad (2)$$

Thus, optimizing domain distribution means to find $S_{i,t+1}$ minimizing the expression

$$\sum_{i=1,n} |Ta_t - Tc_{i,t} S_{i,t+1}| \quad (3)$$

A solution to this optimization is obtained as follows: we first define $S_{i,t+1} = S_{i,t}$ and sort all PEs according to the value $|Ta_t - Tc_{i,t} S_{i,t+1}|$ (see fig. 2). In this way, we find the worst sized domain $S_{\gamma(1),t+1}$ (either oversized or undersized). Then we try to adjust the size of $S_{\gamma(1),t+1}$ by virtually exchanging segments of cells with the neighboring sub-domains so as to make $Tc_{i,t} S_{i,t+1} \approx Ta_t$. The new sub-domains resulting from the motion of the boundary of rectangle $\gamma(1)$ can be computed by a recursive procedure. The same algorithm is run by each processor since all of them have a complete list of the coordinates of the all rectangles. This new domain decomposition is performed four times, for an east, west, north and south boundary motion and the best of these solutions is selected to give $S_{i,t+1}$. However, this optimization may not be possible because: (i) the new partition is more unbalanced than the previous one; (ii) a domain may shrink beyond an acceptable size (e.g. in finite difference schemes, a specific number of neighbor cells are required); in such a case the optimization is considered for $S_{\gamma(next)}$;

When the optimization is profitable, the whole process is repeated (sorting and optimizing S) until a pre-assigned maximum number of steps is performed or until the difference of time between the fastest and slowest PE is less than a given threshold.

Once the new partition is determined, the PEs whose domain has shrunk must send the unused data to the PEs whose domain has increased. Also, the

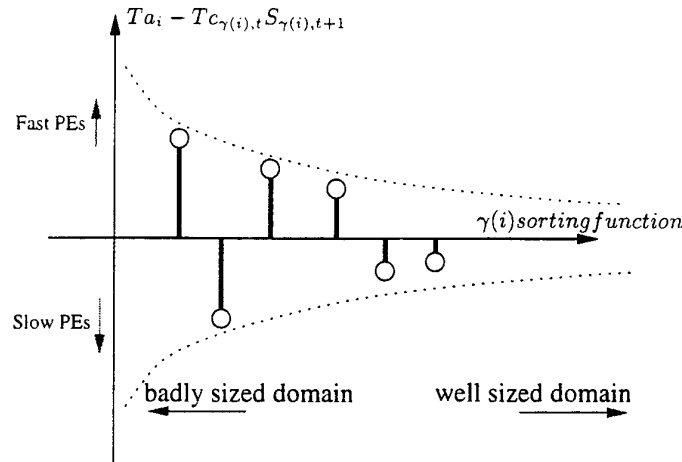


Fig. 2. PE are sorted according to their size.

regular communications between adjacent PEs are affected by the new partition and a new communication pattern has to be generated. Figure 3 shows two configurations, before and after load balancing (upper-left and upper-right partitions, respectively). The data motion problem is solved by superimposing the shape of the new configuration, including ghost cells used for communication, on top of the old one. From the intersections, one can determine which block should be sent and which should be received.

This global dynamic load balancing algorithm is very efficient when used with a restricted amount of PEs (up to 10-16), because its complexity grows very rapidly as the number of PEs increases. It is then well adapted to a coarse grain parallelism.

2.2 The local scheme

For a fine grain parallelism, a local policy is more suitable. The main problem is for a PE to know whether it can move its boundary since, usually, such a modification affects more than the immediate neighbors. In our local strategy, each PE converses several time with its immediate neighbors, in order to let information propagate.

First, each PE compares its last CPU time measurement with those of its adjacent neighbors. The locally slower PEs try to shrink their domains, while the faster ones try to obtain a larger domain. This decision is taken jointly by propagating a request among the PEs concerned by the change (see fig. 4). The algorithm proceeds as follows: let us assume that n PEs have a north/south boundary along the same vertical line. In the worst case, a full column of PEs could be concerned and n would be of the order \sqrt{p} , where p is the total number

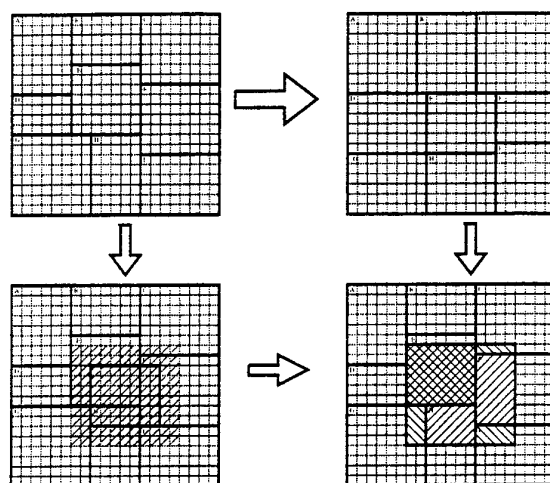


Fig. 3. Use of masks to determine data migration between two configurations (the upper-left to upper-right configuration). We apply to PE E the new partition shape (including ghost border). One block need not migrate, whereas five blocks must be imported from neighbors.

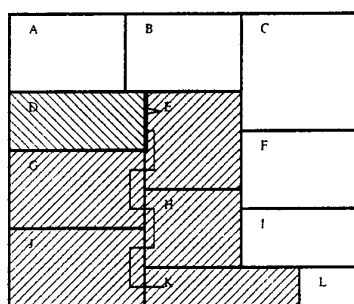


Fig. 4. Taking a coherent decision about moving a boundary is difficult because this affects non-local neighbors: if PE D wants to move its eastern boundary, it will move PE K western one. The decision of how much each boundary will move is made by propagating request messages across the affected region.

of processor. When PE number i wants to move its east boundary, it exchanges with all its adjacent east neighbors a message of size $2 * n - 1$ with, in position i , a request containing the amplitude of the desired motion. The message size $2 * n - 1$ ensures that all PEs along the same vertical boundary can insert in the message the requested motion. Once all slots of the message are filled in (i.e. after $2 * n - 1$ iterations), the processors may take a common decision, which is, in the current implementation, to move the boundary by the average displacement. The same procedure can then be repeated across the east/west boundaries.

When each PE knows in which way it will have to move its boundaries, it must determine if its new neighbors lists resulting from the modified partition has changed (see fig. 5). To obtain this information, it must know the new coordinates of the domains owned by the surrounding PEs. This can be obtained by propagating each PE coordinates a number $\max(m, n)$ of times along the connected sub-domains, where m is the number of PEs sharing the same horizontal boundary. Then, it is possible to reorganize the data using the same technique as discussed in section 2.1, but with a local computation.

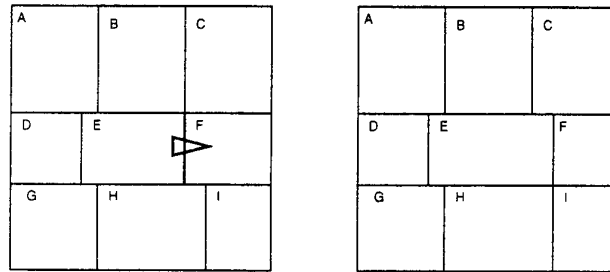


Fig. 5. Due to the east boundary modification, PE E will have to add PE C in its neighbors list.

3 Applications

3.1 Modeling pollution transport

We have parallelized an atmospheric pollution transport application involving 35 chemical species [MOC⁺ed], distributed on a 3D regular grid and implementing more than one hundred chemical equations depending on various atmospheric parameters like wind, temperature and humidity. For designing reasons, the 3D spatial domain is divided in columns along the horizontal plane. After each useful calculation step, a global communication step is done to determine if load balancing is necessary. This kind of compute intensive problem on a small domain (grid dimension is about 50x40 columns) is adapted to test our global load balancing approach.

We have ported the application on our Department workstations which are Sun sparc 4 & 5 and ultra 1 machines. In this case load balancing is critical because there is not any control on each CPU availability. Early in the morning (where little activity is recorded but different architectures are used) our problem can be solved, without load balancing, in 1h13mn. This can easily turn to 2 hours in middle of the day. When turning on our global load balancing algorithm, a run takes only about 42mn, even during the peak activity time. Figure 6 show the time per computation step as a function of the iteration. Without load balancing, we can see a significant variation of performance among the machines. With load balancing, the time spent by each PE is much more homogeneous.

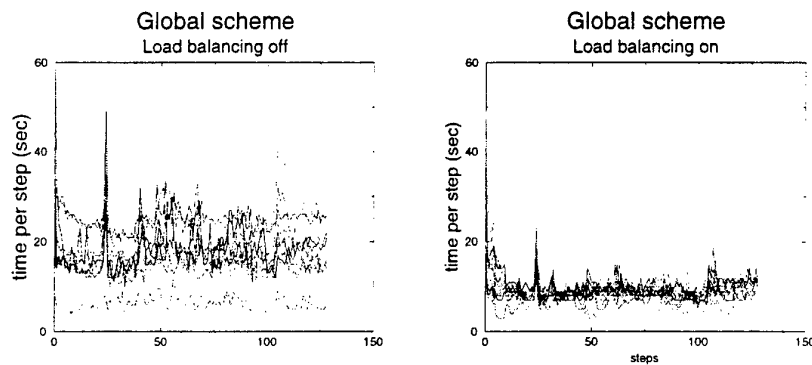


Fig. 6. CPU time per iteration as a function of the iteration, for 16 heterogeneous workstations running the pollution transport code. On the left, the global load balancing algorithm is off and, on the right, it is on.

3.2 Wave propagation

A second application we have parallelized on a cluster of workstations is a wave propagation model for urban environment. This application is based on the lattice Boltzmann method [CLW97] and uses a numerical scheme closed to the so-called TLM method (Transmission Line Matrix). Wave propagation is a important problem when designing a mobile communication network. Figure 7 illustrates the wave intensity pattern predicted by our application in the case where an antenna is located in the middle of an urban area.

This wave propagation model requires 15 floating point operations per grid site and four communications steps (with the east, north, west and south neighbors). Since the computation step is rather light, synchronization of the PE is frequent and this problem may be difficult to parallelize in a efficient way on a workstation cluster with low communication bandwidth and high latency. With

such conditions it seems to be very difficult to obtain a good load balancing without a significant overhead.

This application corresponds to a real-life problem. The simplicity of the numerical scheme makes it a good candidate for an interesting benchmark of a fine grain parallel application. Table 1 shows the performance obtained for a sequential implementation of this code.

CPU	frequency MHz	memory cache	200x200x800 grid (M-flops)	600x600x1600 grid (M-flops)
DEC alpha	150	96KB	6.0	n/a
Sun Sparc4	110	256KB	7.2	5.4
Intel Ppro	200	512KB	17.1	16.2
Sun Ultra1	166	512KB	18.5	11.8
IBM RS6000	66	64KB	24.9	10.7
Intel PII	266	1MB	25.9	22.0
SG RS10000	175	1MB	33.1	22.4

Table 1. Some benchmark results with the best optimized sequential version of our TLM wave propagation model. Note that all these figures are obtained for simulations on square domains, without buildings. Two tests were done: the first one corresponds to running the application on a grid of size 200x200 for 800 iterations; the second test is for a 800x800 grid during 1600 iterations. Due to cache missed augmentation, performances decrease when domain sizes increases. RS10000 results may be pessimistic because of a possible overload of the machine when benchmark were done.

The parallelized version of the TLM simulations has been run on a 32-PC cluster running Linux and PVM [BDG⁺91], each node being a Pentium II 266 MHz with 64MB of memory. All PCs are interconnected using two 18-entry switches, with fast-Internet links. A Sun Sparc 5 with NFS is used as file server. The advantage of a such solution is a very low price (about 60'000 dollars) with pretty good performances, a full control on each node since the machine can be dedicated to only parallel applications.

Load imbalances are due to the fact that, during the first iterations, only the processor containing the antenna site must perform useful computations. As the wave front propagates, all processors become active. The presence of buildings may be another reason to produce an uneven load distribution among the processors because no computation is needed on building sites.

Due to some specific optimization techniques, only valid in the sequential version of the code (memory moves replaced by alias pointers), the parallel version is slower by a factor 2, even if run on a single processor: on a 200 × 200 domain, 800 iterations take 16.2 seconds to complete for the sequential code and 32.8 for the parallel one. Therefore, we do not expect an efficiency larger than 50% for the parallel implementation. However, as the problem size increases (e.g. 1600 × 1600 during 4800 iterations), the code no longer runs on a single PE without swapping. This makes the parallel implementation very effective



Fig. 7. Processing wave propagation in urban environment with 16 workstations, using our load balancing scheme. The rectangles show the block of data assigned to each processor.

For this problem size, the parallel code runs in 1026 sec, without load balancing and using 16 free PEs. In fig. 8 we show the evolution of the time needed to perform chunk of computation corresponding to 16 propagation steps. After an initial stage (where some Pes have not yet been reached by the wave), the computation time becomes uniform and each chunk of 16 consecutive iterations takes about 4.5 seconds.

In order to test our local load balancing algorithm we launch on a node a program aimed at perturbing the computation. This program, given in the appendix, uses lot of CPU. As a consequence of this extra load, the TLM execution time increases to 4976 sec and the infected node performs the 16 steps in about 25 sec, thus slowing down the whole system.

Under the same condition, the local load balancing algorithms, modify the domain partition and the total CPU time come down to 2243 sec. This effect is illustrated in fig 8 is erased. However this still is slower than the version without perturbation. This factor is due to poor stability of local-loadbalancing which will improved in further work.

Figure 9 shows the speedup obtained with and without load balancing. Comparison is made with the most optimized sequential version [Lut98].

4 Conclusion

From these results we conclude that it is perfectly possible to use clusters of workstations (or PCs) instead of heavy, expensive and less convenient mainframes. With our approach we could obtain satisfactory results in terms of speedup and load balancing, at very low cost.

Some parts of our load balancing algorithms are still heuristically determined and the worst case complexity of the overhead. The convergence time and stability of the partitioning procedure must still be investigated. However, the results

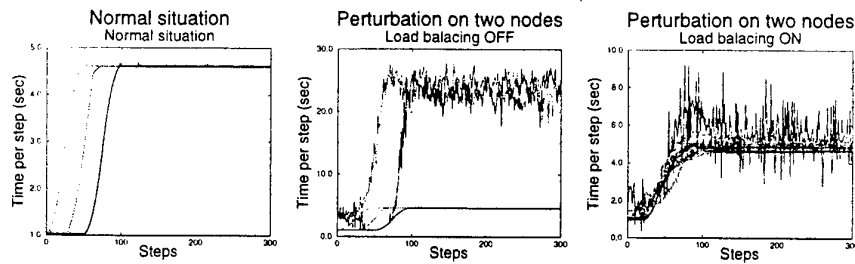


Fig. 8. Evolution of the CPU time needed to perform 16 iterations of the TLM problem, with 16 Pentium II-266. On the left, all PEs are dedicated to the computation and a balanced situation is observed. The middle plot corresponds to a situation where 2 nodes are disturbed with a intensive CPU consumer program. A clear load unbalance is observed, unless our local algorithm is turned on, as shown on the left.

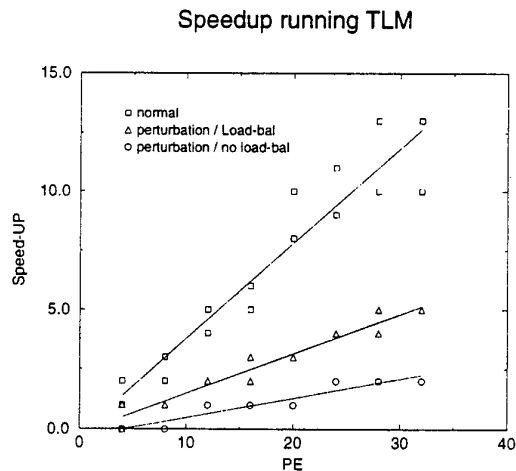


Fig. 9. Speedup computed for problems of both sizes 1600×1600 and 800×800 run for 4800 iterations. The best sequential time, estimated to 118 minutes, is interpolated from the timing of the 800 problem size (since the large grid size would cause swapping on one single PE). Performances of the load-balancing algorithm might improved by decreasing the unstability seen in fig 8.

discussed here show clearly the benefit of our load balancing strategies when the PEs are heterogeneous or load in an unpredictable way by other jobs.

References

- [BDG⁺91] Beguelin, Dongarra, Geist, Manchek, and Sunderam. A user's guide to pvm 'parallel virtual machine'. *ORNTNLTM-11826*, 1991.
- [CLW97] B. Chopard, P.O. Luthi, and Jean-Frédéric Wagen. A lattice boltzmann method for wave propagation in urban microcells. *IEEE Proceedings - Microwaves, Antennas and Propagation*, 144:251-255, 1997.
- [Fos95] Ian Foster. *Designing and building Parallel programs*. Addison-Wesley, 1995.
- [KW] Kuchen and Wagener. Comparaison of dynamic load balancing strategie. <http://fiachra.ucd.ie/~david/loadbalancingpapers.html>.
- [Lut98] Luthi. *lattice wave automata*. PhD thesis, University of Geneva, 1998.
- [MOC⁺ed] M. Martin, O. Oberson, B. Chopard, F. Mueller, and A. Clappier. Atmospheric pollution transport: The parallelization of a transport & chemistry code. *Atmospheric Environment*, submitted.

This research is supported by the Swiss National Science Foundation.

Appendix: Slowing down a node

This program is used to slow down a Unix system: it starts to fork and produces *n* copies of itself. Each copy allocates a big block of memory and accesses it randomly. When the parent process is killed with the `kill -USR1 pid` command, all child processes are killed too. Note that the CPU overload created is not constant because swapping events may randomly appear.

```
/*
 * little program used to slow down a UNIX station
 */
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>

/* memory used per process (1Mo) */
#define BLOCK 0x100000

/* some global datas */
int* pids;
int nb_childs;

/* to kill all processes */
void kill_childs()
{ int i;
  for (i=0;i<nb_childs;i++) kill(pids[i],SIGUSR2);
```

```

    printf("\nOk, I'm dead.\n");
    exit(0);
}

void main(char argc, char** argv)
{
    double* datas;
    int i;
    int nb_el = BLOCK/sizeof(double);
    int pid = 1;
    signal(SIGUSR1, kill_childs); /* to kill all processes */

    if (argc!=2) {printf ("Usage : %s nb_processes\n",argv[0]);exit(0);}
    nb_childs = atoi(argv[1])-1; /* get nb processes from command line*/

    printf ("Starting with %d processes\n",nb_childs+1);
    printf (" to kill me use the \"kill -USR1 %d\" command\n",getpid());

    pids = (int*) malloc (nb_childs*sizeof(int)); /* child ids reminder */
    datas = (double*) calloc(nb_el,sizeof(double)); /* memory allocation */

    for (i=0;i<nb_childs;i++) /* creating child */
        if (pid) {pid=fork();pids[i]=pid;}

    srand (getpid()); /* pseudo-random-number generator init */
    while (1){datas[rand()%nb_el]++;} /* random accesses in memory */
}

```